Theory of Computation

Mohamed Kobeissi

February 11, 2019

What is this course about ?

Can a computer solve answer if any Mathematical sentence is true or false ?

What can be done by a computer and what cannot ?

Computability and complexity: what is the difference ?

Contents

1) Alphabets, strings, languages. Finite automata (DFA, NFA). Design of automata, NFA conversion to DFA. Regular expressions and regular languages. minimization of DFA. Pumping Lemma
2) Context-free languages. Contect-free grammar. CFG for regular languages. CFG for non-regular languages. Pushdown Automata

Textbook: Introduction to the Theory of computation, third edition, Michael Sipser, Cengage Learning

mohamed.kobeissi@ul.edu.lb

Alphabets, strings, languages

Alphabets

An alphabet, denoted Σ, is a finite set of symbols

- The Binary Alphabet $\{0,1\}$
- The English Alphabet $\{a, b, \ldots, z\}$
- The ASCII alphabet consisting of 128 symbols

Strings

- A string is a finite sequence of symbols from an alphabet written in juxtaposition
- The string containing no symbols is the empty string, denoted by λ (also ε)
- The length of a string is its length as a sequence. For example, the length of the string *ababba*, denoted by $|ababba|$, is 6, and $|\lambda| = 0$
- The set of all strings over alphabet Σ is denoted by Σ^*

Operations on strings

- **Concatenation:** The concatenation of two strings $u = a_1 a_2 \dots a_n$ and $v = b_1 b_2 \dots b_m$, denoted $u \circ v$ (or simply uv), is the string $w = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$

  - $u = 0111001$ and $v = 0111001$, then $uv = 01110010111001$

  - $x = abb$ et $y = aa$, then $xy = abbaa$

The following properties are obvious:

- $\lambda w = w\lambda = w$
- $|uv| = |u| + |v|$

- $\lambda 011110 = 011110$

- In general, uv is different from vu; the concatenation is not commutative

if $u = 10$ and $v = 1$, then $uv = 101$ and $vu = 110$

- **Reverse:** The reverse of a string $u = a_1 a_2 \ldots a_n$, denoted u^R, is defined by $u^R = a_n a_{n-1} \ldots a_1$
  for example if $v = 01100$, then $v^R = 00110$

  - $(uv)^R = v^R u^R$

  - a string w is a palindrome iff $w = w^R$

  - *abba* is a palindrome, while 01100 is not

- **Power:** The power of a string u, denoted u^n, is defined by

$$u^n = \underbrace{uuu \circ \ldots \circ u}_{n \ \ times}$$

- u^n is the concatenation of u with itself n times; note that $u^0 = \lambda$

- $u = 011$, then $u^3 = 011011011$

Show that u is a palindrome iff u^n is a palindrome

Let a, b, c and d be 4 strings on the same alphabet with $ab = cd$, is $a = c$ and $b = d$? prove or give a counter example
- Give a necessary and sufficient condition for the above to be true

- **substring:** A substring is a consecutive sequence of alphabet from a string w

  - ab, bba are substrings of $abbab$, aba is not

- **Prefix and suffix:** A string $w = a_1 a_2 \ldots a_n$ may be seen as the concatenation of two strings $u = a_1 a_2 \ldots a_i$ and $v = a_{i+1} a_{i+2} \ldots a_n$; these two strings are respectively called prefix and suffix of w

  - If $w = abbab$, then $w = uv$ with u and v are listed in the following table:

prefix	suffix
λ	$abbab$
a	$bbab$
ab	bab
abb	ab
$abba$	b
$abbab$	λ

Languages

- A language L is a set of strings defined over an alphabet Σ
- In terms of sets, a language is a subset of Σ^*

Basic Operations on Languages

The basic operations, like union, intersection and complement are defined as "usual" :

- Union: $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$
- Complement: $\overline{L} = \{w \mid w \notin L\}$
  (unless further notice, the complement is considered with respect to Σ^*)

Special operations for Languages

Following are some special operations for languages:

- Concatenation: $L_1 \circ L_2 = \{w = xy \mid x \in L_1, y \in L_2\}$
- Reverse: The reverse of a language L is defined by

$$L^R = \{w^R; w \in L\}$$

Power

- Power: The power of a langauge L is defined

$$L^n = L \circ L \circ L \ldots \circ L$$

(Note that $L^0 = \lambda$)

Thus, the language L^n, a language consisting of strings of length n, can be built inductively as $L \circ L^{n-1}$

Note that $L \circ K$ may be denoted LK for simplicity

Kleene closure

- The Kleene Closure of a language L, denoted L^*, is defined by

$$L^* = \bigcup_{i \in \mathbb{N}} L^i$$

- The positive closure, denoted L^+, is defined by

$$L^+ = L^* \setminus \{\lambda\}$$

- If $L_1 = \{a, ab, aaaa\}$ and $L_2 = \{\lambda, bb, ab\}$,
  - $L_1 \circ L_2 = \{a, abb, aab, ab, abbb, abab, aaaa, aaaabb, aaaaab\}$
  - $L_1 \cup L_2 = \{\lambda, a, ab, bb, aaaa\}$
  - $L_1 \cap L_2 = \{ab\}$
  - $L_2^R = \{\lambda, bb, ba\}$
  - $(L_1)^* = \{\lambda, a, ab, aaaa, aa, aab, aaaaa, aba, abab, abaaaa, aaaaab, aaaaaaaa, \ldots\}$

Show that the language

$$L = (1 \cup 01)^*(0 \cup \lambda)$$

consists of all words on $\{0, 1\}^*$ with no consecutive zero

Regular Expressions

Regular Expressions

A regular expression r is a pattern that matches a set of strings. For example, the regular expression $ab(a^*)$ matches the strings ab, and aba, and $abaaa$, and any string that starts with ab followed by any number of a's. The $*$ (called "Kleene star") indicates that the pattern should be repeated 0 or more times. The set of strings matched by an expression r is called the language of the expression and is denoted $L(r)$

Basic Regular Expressions

In general, a regular expression is any of the following:

- a single character $a \in \Sigma$. The language of the expression a is just $\{a\}$
- The expression $\emptyset$ which matches no strings
- The expression λ which matches only the empty string

Regular expressions are defined recursively as follows: if r_1 and r_2 are regular expressions, then so are

- $r_1 + r_2$ where r_1 and r_2 are regular expressions. This is called the union of r_1 and r_2. It matches any string matched by either r_1 or r_2: $L(r_1 + r_2) = L(r_1) \cup L(r_2)$

- the concatenation of two regular expressions $r_1 r_2$, which matches any string formed by concatenating a string in $L(r_1)$ with a string in $L(r_2)$. Formally, $L(r_1 r_2) = L(r_1)L(r_2) = \{xy \mid x \in L(r_1) \text{ and } y \in L(r_2)\}$
- r^* where r is any regular expression. As described above, $L(r^*) = \{x_1 x_2 x_3 \cdots x_n \mid x_i \in L(r)\}$

The value of a regular expression is a language

Example

For example

$$L\big((a + bc)^*\big) = \{\lambda, a, bc, aa, abc, bca, bcbc, aabc, \ldots\}$$

1) $r = (a+b)a^*$, then

$$\begin{aligned} L((a+b)a^*) &= L((a+b))L(a^*) \\ &= \{L(a) \cup L(b)\}L(a^*) \\ &= \{\{a\} \cup \{b\}\}(\{a\})^* \\ &= \{a \cup b\}\{\lambda, a, aa, aaa, \ldots\} \\ &= \{a, aa, aaa, \ldots, ba, baa, baaa \ldots\} \end{aligned}$$

We could say $L((a+b)a^*) = a^+ \cup ba^*$

2) $r = (a + b)^*(a + bb)$

3) $r = (aa)^*(bb)^*b$

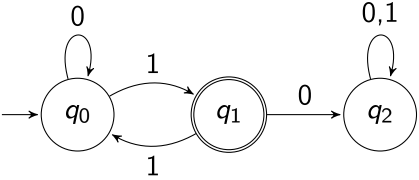
4) $r = (a+b)^* aa(a+b)^*$

5) Let $r = (1+01)^*(0+\lambda)$. Prove that $L(r) =$ {all words with no consecutive zero}

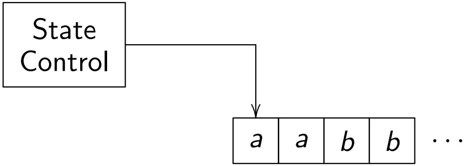
Finite Automata and regular grammar

Automata

A finite automata is an abstract machine that starts in an initial state, and repeats some task until it ends up in a state. An automata may have 0 or more accepting (or final) states but only a single start state. When using an automata to implement a language, the validity of a string is determined by whether or not the ending state of the automata after parsing is an accepting state



The following figure is a schematic representation of a finite automaton. The control represents the states and transition function, the tape contains the input string, and the arrow represents the input head, pointing at the next input symbol to be read

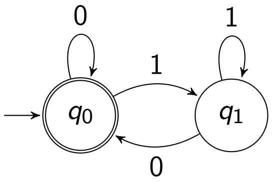


Deterministic Finite Automata (DFA)

A DFA cannot have more than one transition leaving a state on the same symbol. A DFA will always produce the exact same path for any given string. A DFA formally is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

- Q: Finite, non-empty set of states
- Σ: Input alphabet
- δ: A transition function $\delta : Q \times \Sigma \rightarrow Q$
- q_0: A initial state $q_0 \in Q$
- F: A set of accepting (final) states $F \subset Q$

The automata



can be defined formally by the 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1\}, \Sigma = \{0, 1\}, q_0, F = \{q_0\}$, and the transition function δ defined by:

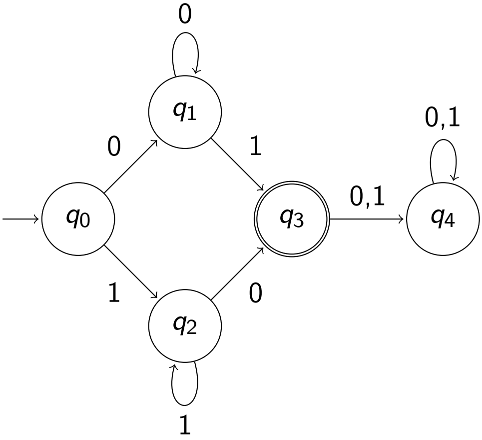
δ	0	1
q_0	q_0	q_1
q_1	q_0	q_1

Language of a DFA

- A string w is accepted by a DFA if the computation of w ends up in an accepting state of this automata
- The language $L(M)$, accepted by a DFA M is the set of all strings accepted by M

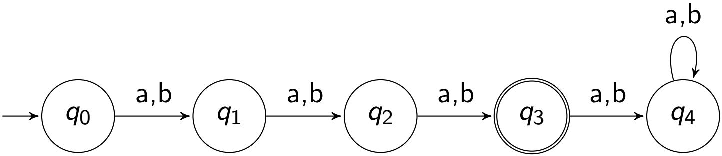
Example 1

Find all the strings accepted by the following automata (guess!!)



Example 2

What about this automata ?

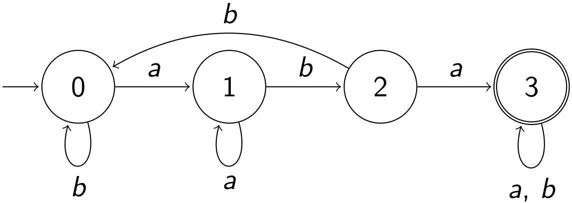


Deduce an automata that accepts the language L defined by:

$$L = \{w; |w| = 3k\ ,\ k \leq 0\}$$

Example 3

And this one ?



Two important questions arises:

1) Given a certain automata, what is the language accepted by this automata
2) Given a language, is there is an automata that recognized it

Designing DFA

1) Draw deterministic finite automata that accepts all strings on $\{0, 1\}^*$, having an even number of 1

2) Draw deterministic finite automata that accepts all strings on $\{a, b\}^*$ and containing abb as substring

3) Draw deterministic finite automata that accepts all strings on $\{a, b\}^*$ that ends with abb

4) Give deterministic finite automata that accept all the strings on $\{0,1\}^*$ except the two strings 11 and 111

5) $L = \{w \in \{0,1\}^*; w \text{ contains } 10^n1 \text{ as substring, for some } n \text{ odd}\}$

6) Let $\Sigma = \{a, b, c\}$, and L the language defined by:

$$L = \{w \in \Sigma^*;\ |w|_a \equiv |w|_c + 1 \mod (3)\}$$

where $|w|_x$ *is the occurrence of the string* x *in* w

Give DFA that recognizes the language L

Regular language

Definition

A language is regular if its accepted by a certain DFA

Properties of regular languages

Proposition

If L is regular, then $\overline{L}$ is regular

As L is regular, there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$. The automata $\overline{M} = (Q, \Sigma, \delta, q_0, Q \setminus F)$ accepts $\overline{L}$ (why ??)

Formally, if an automata accepts a language L, then interchanging the accepting states with non accepting states yields an automata that accepts $\overline{L}$

Proposition

Let L_1 and L_2 be two regular languages, then $L_1 \cup L_2$ is regular

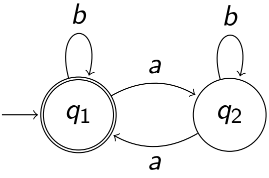
As L_1 and L_2 are regular, there exist two automata $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ that recognize L_1 and L_2 respectively

Define an automata $M_3 = (Q_3, \Sigma, \delta_3, q_{03}, F_3)$ as follows:

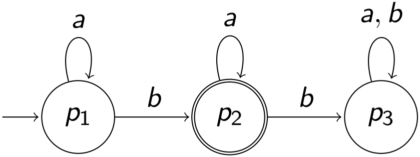
- $Q_3 = Q_1 \times Q_2$; the states of M_3 are the cartesian product of the states of Q_1 and Q_2
- $q_{03} = (q_{01}, q_{02})$
- $\delta_3((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $F_3 = \{(q_1, q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2\}$

M_3 recognizes the language $L_1 \cup L_2$ (why ?)

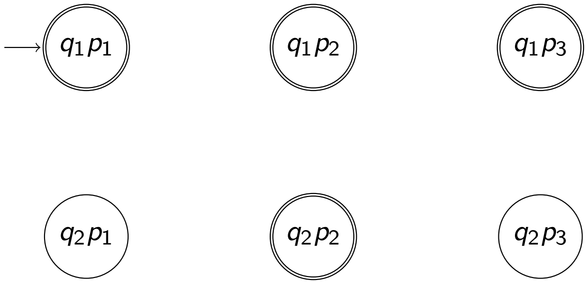
Give a DFA that accepts the $L(M_1) \cup L(M_2)$



M_1



M_2


$q_1 p_1$
$q_1 p_2$
$q_1 p_3$
$q_2 p_1$
$q_2 p_2$
$q_2 p_3$

Intersection of regular languages

Is the intersection of two regular languages also regular ?

Grammars

A grammar is a 4-tuple (V, Σ, R, S), where

- V is a finite set called the **variables**
- Σ is a finite set, disjoint form V, called the **terminals**
- R is a set of **rules**, with each rule being a variable and a string of variables and terminals, and
- $S \in V$ is the start variable

- If u, v, and w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that uAv **yields** uwv
- We say that u **derives** v , written $u \overset{*}{\Rightarrow} v$ if $u = v$, or if a sequence $u_1, u_2, \ldots, u_k$, exists for $k \geq 0$ and

  $$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$$

  Such a sequence is called a **derivation** or **parse**, and discovering the derivation is called **parsing**

Language of a grammar

- The language generated by a grammar is the set of all strings of terminals produced from S using rules (or productions) as substitutions
- The language of a grammar is defined by:

$$\{w \in \Sigma^*; S \overset{*}{\Rightarrow} w\}$$

Regular Grammars

A grammar is **regular** if all the rules have one of the following forms:

$$A \to aB \qquad (1)$$

$$A \to a \qquad (2)$$

$$A \to \lambda \qquad (3)$$

where A, B are in V and $a \in \Sigma$
Rule (1) may be extended to $A \to wB$ where w is a string

Example:

$G = (\{S\}, \{x, y, z\}, R, S)$, where the productions of R are:

$$S \to xS$$
$$S \to y$$
$$S \to z$$

The above grammar can be equivalently defined by:

$$S \to xS|y|z$$

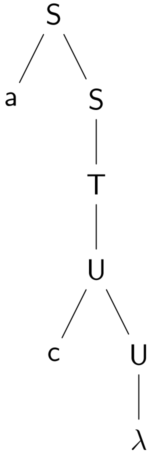
Example:

For the grammar

$$
\begin{aligned}
S &\to aS|T \\
T &\to bT|U \\
U &\to cU|\lambda
\end{aligned}
$$

we have the derivation
$S \implies aS \implies aT \implies aU \implies acU \implies ac$

The parse tree for the derivation is:



Proposition: A language is regular iff it is generated by a regular grammar

Example:

Find the language defined by the grammar G below:

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow \lambda \end{aligned}$$

Note that the grammar can be equivalently defined by:

$$S \rightarrow aS|\lambda$$

Example:

The grammar

$$S \rightarrow aS|bS|\lambda$$

generates all the words in $\{a, b\}^*$

Grammars for regular languages

Constructing a grammar for a language that happens to be regular is easy if you can first construct a DFA for that language. You can convert any DFA into an equivalent regular grammar as follows. Make a variable R_i for each state q_i of the DFA. Add the rule $R_i \to aR_j$ to the grammar if $\delta(q_i, a) = q_j$ is a transition in the DFA. Add the rule $R_i \to \lambda$ if q_i is an accept state of the DFA. Make R_0 the start variable of the grammar, where q_0 is the start state of the machine

Find a regular grammar for the language

$$L = \{w \in \{a, b\}^* \ ; \ \text{every } a \text{ in } w \text{ is followed by at least one } b\}$$

Example:

- Find the language defined by the grammar G

$$
\begin{aligned}
S &\rightarrow XaaX \\
X &\rightarrow aX|bX|\lambda
\end{aligned}
$$

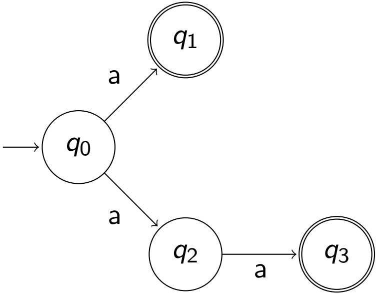
Then give a derivation of the string *ababaaaba*

- Is the grammar G regular ? if not give a regular grammar whose language is $L(G)$

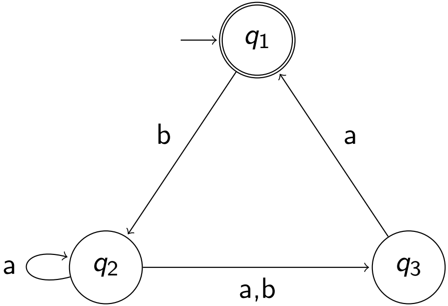
Nondeterministic Finite Automata (NFA)

An NFA can have multiple multiple paths leaving a state encoded with the same symbol. An NFA will yield many possible paths for a given string. Formally, an NFA is a 5-tuple $(Q, \Sigma, \Delta, q_0, F)$ with $\Delta : Q \times \Sigma \to P(Q)$ where $P(Q)$ is the powerset of Q

Example 1

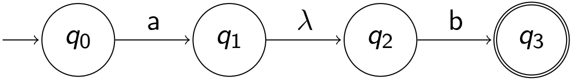


Example 2

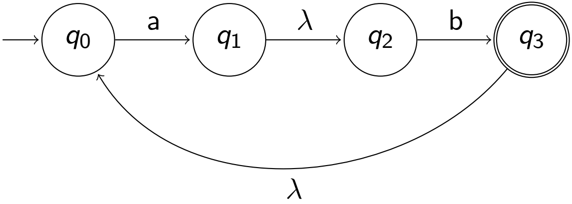


Example 3

An NFA with λ-transition



What about this NFA ?



i) Give a DFA that accepts the language $L = \{ab\}$

ii) Give DFA that accepts L^+, and a DFA for L^*

Language of an NFA

- A string w is accepted by an NFA if at least one path of the computation tree of w ends up in an accepting state of this automata
- The language $L(N)$, accepted by an NFA N is the set of all strings accepted by N

Designing NFA

1) Draw an NFA automata that accepts the language

$$L = \{w \in \{0,1\}^*; \text{every odd position of w is 1}\}$$

2) Draw an NFA automata that accepts the language

$$L = \{w \in \{a, b\}^*; \text{the third symbol from the end of w is a}\}$$

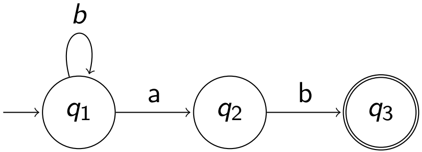
NFA with one single final state

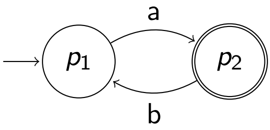
Every NFA is equivalent to an NFA with one single accepting state (how?)

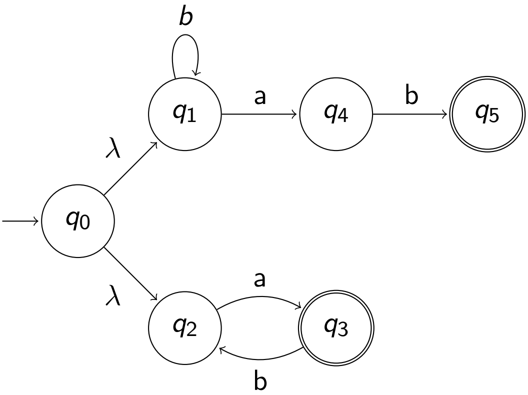
NFA union

Constructing an NFA to recognize the union is much easier: we can simply create a new start state with lambda transitions to the start states of the two original machines

Example



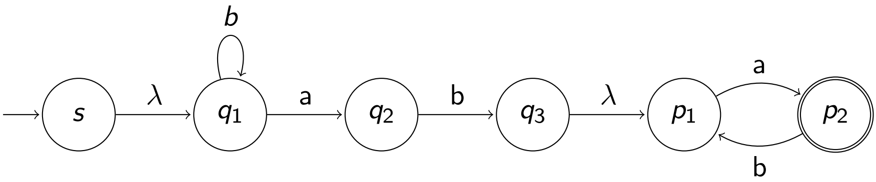



b
a
b
λ
λ
a
b
q1
q4
q5
q0
q2
q3

NFA concatenation

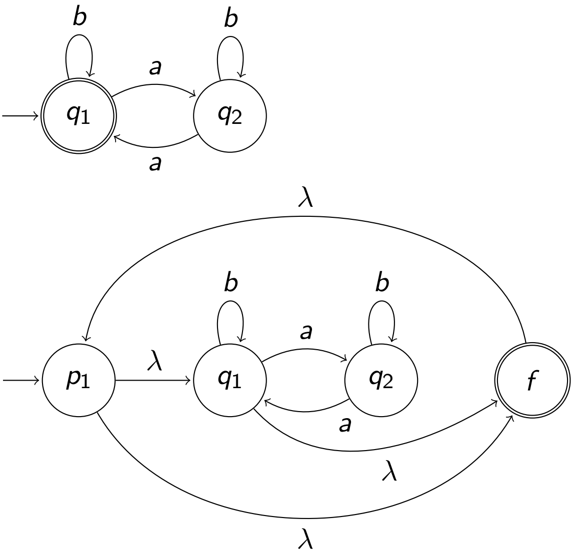
Suppose that N_1 and N_2 are two NFA that recognize the languages L_1 and L_2 respectively. Suppose further that N_1 and N_2 have one single accepting state each. To construct an NFA that recognizes the language $L_1 L_2$ simply add a lambda transition from the final state of N_1 to the initial state of N_2, the resulting automata will do (why?)

Example



Kleene closure

Suppose N is an NFA that accepts a language L (assume N that one accepting state). To construct an NFA that accepts L^*, simply add a new initial state s with a lambda transition to the initial state of N, and add one accepting state f with lambda transition form the accepting state of N to f. Add also lambda transitions form s to f and from f to s


b
b
a
q_1
q_2
a
λ
b
b
a
p_1
λ
q_1
q_2
f
a
λ
λ

Equivalence of Automata

Two automata are equivalent iff they recognize the same language

NFA to DFA conversion

Proposition

Every NFA can be converted to an equivalent DFA

Lemma

A language is regular iff its accepted by a certain automata (DFA or NFA)

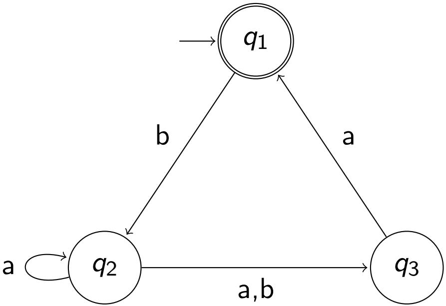
Consider the NFA $N = (Q_N, \Sigma, \delta_N, q_{0N}, F_N)$. Define the DFA $M = (Q_M, \Sigma, \delta_M, q_{0M}, F_M)$ as follows:

- $Q_M = \mathcal{P}(Q_N)$
- $q_{0M} = \widehat{\lambda}_N(q_{0N})$
- $\delta_M(S, a) = \bigcup_{q \in S} \widehat{\lambda}_N(q, a)$
- $F_M = \{S \in Q_M \mid \exists q \in S \text{ such that } q \in F_N\}$

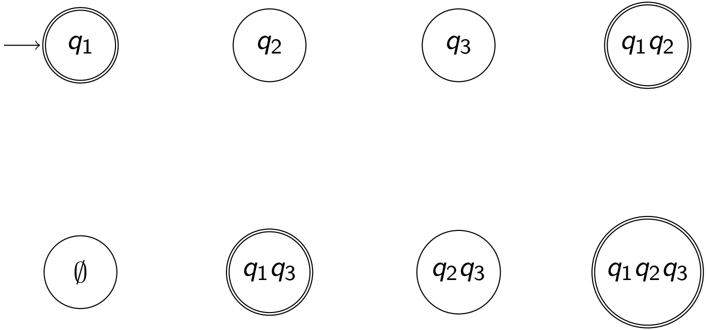
M recognizes the same language as N (why ?)

Example 5

The transition function δ_M is given for the NFA below (NFA example 2))

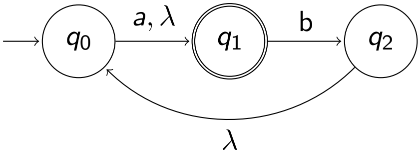


δ_M	a	b
q_1	$\emptyset$	q_2
q_2	$q_2 q_3$	q_3
q_3	q_1	$\emptyset$


q_1
q_2
q_3
$q_1 q_2$
$\emptyset$
$q_1 q_3$
$q_2 q_3$
$q_1 q_2 q_3$

Example 6

Convert the below NFA into an equivalent DFA



Regular Expressions and regular languages

Proposition

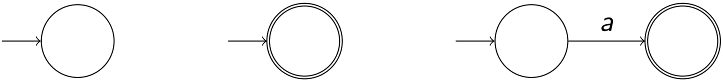
Every regular expression r is recognized by a certain automata

Lemma

A language L is called regular if there is some regular expression r with $L(r) = L$

Regular Expressions to NFA conversion

The following NFA's recognize the "basic" regular expression:



$L = \{\emptyset\}$ $L = \{\lambda\}$ $L = \{a\}$

Automata for "non-basic" regular expressions can be deduced from the above knowing that the class of regular languages is closed under the regular operations

Example 1

Give finite automata that recognizes the expression

$$(ab + a)^*$$

Example 2

Give finite automata that recognizes the expression

$$a^{+}b(ab + a^{*})$$

Example 3

Deduce finite automaton that recognize the expressions

$$(a^+b(ab + a^*))^*$$

and

$$a^+b((ab + a^*))^*$$

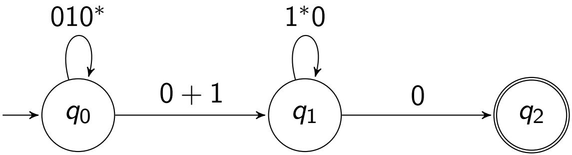
Regular languages to regular expressions

Proposition

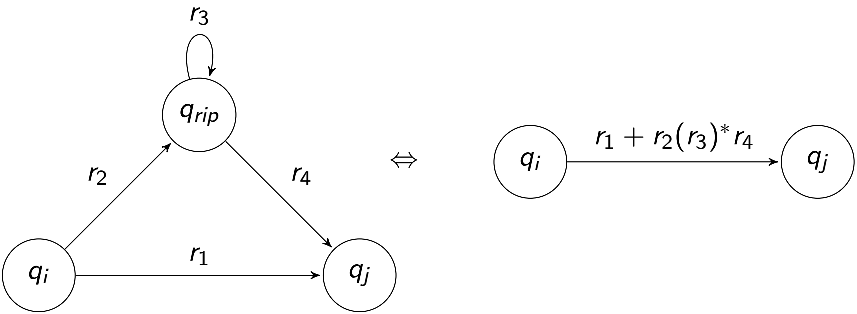
Any regular language can be described by a regular expression

Generalized non deterministic finite automata (GNFA)

A generalized non deterministic finite automata (GNFA) is an automata were transition are regular expressions instead of alphabets



Equivalence of GNFA's



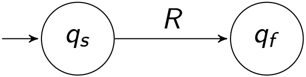
The two GNFA above are equivalent in the sense that they both accept the same regular expressions

DFA conversion to regular expressions

To convert a DFA to a regular expression, transform the DFA into GNFA as follows: start by adding two states q_s and q_f. Add a λ-transition from q_s to the initial state, and λ-transitions from each accepting state to q_f. A multiple transition are replaced by expressions as shown in the figure

$$q_i \xrightarrow{0,1} q_j \quad \Leftrightarrow \quad q_i \xrightarrow{0+1} q_j$$

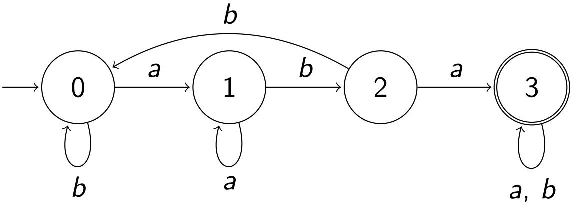
Start ripping (removing) states one after the other and keeping GNFA's equivalent. We finally obtain the following:



The expression R is the regular expression describing the language recognized by the automata

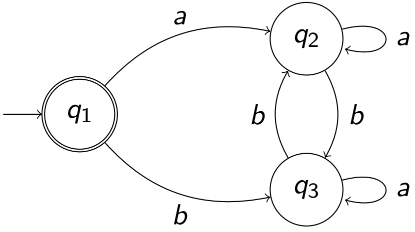
a) Removing states in different order may result in different expressions, but same language !!
b) Applying the GNFA method to an NFA with λ-transition may not give the exact regular expression

Find a regular expression for the language of the following automata:

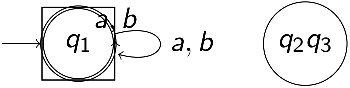


Minimization of DFA

The following DFA clearly recognizes the language $\{\lambda\}$:



In a sense, the states q_2 and q_3 are equivalent: if we start processing a string x in either of them, we will always get the same answer. So we can join them together into a single big state:



We can generalize this idea. Let $\sim$ be the equivalence relation on Q defined by defined by

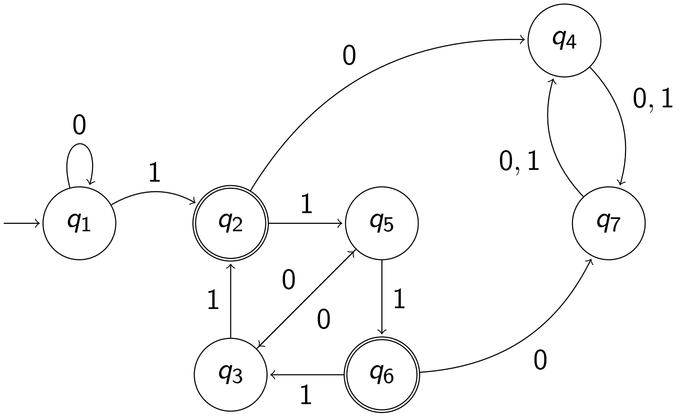
$$q_1 \sim q_2 \text{ iff } \forall x \in \Sigma^*, \widehat{\delta}(q_1, x) \in A \Longleftrightarrow \widehat{\delta}(q_2, x) \in A$$

This formalizes the idea that if we start processing x in q_1 or in q_2, we will always get the same answer

If we know $\sim$, we can construct an equivalent machine M_{min} as follows:

- The states Q_{min} are equivalence classes of states of M: $Q_{min} = Q_M / \sim$
- The accepting states of Q_{min} are the equivalence classes of accepting states of M. Note that if $q_1 \in A_M$ and $q_2 \sim q_1$ then $q_2 \in A_M$
- The initial state of Q_{min} is just $[q_{0M}]$
- The transition function δ_{min} is given by $\delta_{min}([q], a) = [\delta_M(q, a)]$

Find an equivalent minimal DFA for the following automata



The Pumping Lemma

Example

Automata are powerful in the sense that they can recognize any regular language. Do they recognize any type of languages ?
Consider the language $L = \{x \in \{a, b\}^* \mid \#a(x) = \#b(x)\}$

Assume there is a machine automata M that recognizes L. Since Q_M is finite, there are only n states

The string $x = a^{n+1}b^{n+1}$ should be accepted. While processing the $n+1$ a's, the machine must hit the same state q twice. That means that we can split up x:

$$x = \underbrace{aaa\cdots}_{w}\underbrace{\cdots}_{y}\underbrace{\cdots abbb\cdots b}_{z}$$

so that the processing of y starts and ends in q

This means that the string wz is also accepted by M, but $wz \notin L$.

So we have a contradiction

This is an example of a general technique called the pumping lemma

Lemma

If L is a regular language, then there is a pumping length p where, if $w \in L$ and $|w| \geq p$, then w may be divided into three pieces $w = xyz$ satisfying the following:

a) $xy^iz \in L$ *for each* $i \geq 0$

b) $|y| > 0$

c) $|xy| \leq p$

Proof.

□

There are two methods you could use to prove the languages are not regular. The first method is to use the pumping lemma and proof by contradiction. The second method is to use the closure properties of regular languages, known regular languages, and proof by contradiction

The steps to prove that a language is non-regular:

a) Assume for the sake of contradiction that L is regular
b) Since L is regular, there exists a p where for any string $w \in L$ with length $|w| \geq p$, we may divide w into $w = xyz$ such that the conditions of the pumping lemma hold
c) Choose a string $w \in L$ where $|w| \geq p$
d) Show that this string can not be divided into $w = xyz$ such that all of the pumping lemma conditions hold
e) This is a contradiction of the pumping lemma, therefore L is not regular

Example

Prove that $L_1 = \{a^n b^n \mid n \geq 0\}$ is not regular

Note that $\{a^n b^n\} \subset \{a^* b^*\}$, and that $\{a^* b^*\}$ is regular. Thus we just proved that a subset of a regular language is not necessarily regular !!

Example

Prove that $L_2 = \{w \in \{a, b\}^*; |w|_a = |w|_b\}$ is not regular method 1: by pumping lemma (exercise)

method 2: Assume that L_2 is regular, then $L_2 \cap \{a^* b^*\} = \{a^n b^n\}$ is regular, contradiction! hence, L_2 is not regular

Example

Prove that $L_3 = \{0^{2^n} \mid n \geq 0\}$ is not regular

Assume for the sake of contradiction that L_3 is regular, and let p be the pumping length
Let $w = 0^{2^p}$. Clearly $|w| = 2^p \geq p$, then by the pumping lemma we can split w into $w = xyz$ such that $|xy| \leq p$. But $|xy^2z| = 2^p + p < 2^{p+1}$. Therefore $xy^2z \notin L_3$, and then L is not regular

Example

Prove that $L_4 = \{w \in \{a, b\}^*; |w|_a - |w|_b \textit{ is a prime number}\}$ is not regular