CS 301

Lecture 08 – Regular languages recap

Stephen Checkoway

February 12, 2018

Symbols, alphabets, strings, and languages

1. Alphabets are sets of symbols
2. Strings over an alphabet are sequences of symbols from the alphabet
3. Languages over an alphabet are sets strings over the alphabet

Question 1

Can an alphabet contain zero symbols?

Question 1

Can an alphabet contain zero symbols?

No. Alphabets must have at least one symbol

Question 2

Can an alphabet contain infinitely many symbols?

Question 2

Can an alphabet contain infinitely many symbols?
No. Alphabets must be finite

Question 3

Can a string contain zero symbols?

Question 3

Can a string contain zero symbols?

Yes. ε is a perfectly reasonable string

Question 4

Can a string contain infinitely many symbols?

Question 4

Can a string contain infinitely many symbols?

No. Strings must have finite length

Question 5

Can a language contain zero strings?

Question 5

Can a language contain zero strings?

Yes. $\varnothing$ is the empty language

Question 6

Can a language contain infinitely many strings?

Question 6

Can a language contain infinitely many strings?

Yes. Most languages contain infinitely many strings.

(For a given alphabet, there are countably-many finite languages but uncountably-many nonfinite languages)

Deterministic finite automata

DFAs are five-tuples $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is the set of states
- Σ is the alphabet
- δ is the transition function
- q_0 is the start state
- F is the set of accepting states

Question 7

Can Q be the empty set?

Question 7

Can Q be the empty set?

No. Every DFA contains at least a start state q_0

Question 8

Can Q contain infinitely many states?

Question 8

Can Q contain infinitely many states?

No. These are *finite* automata

Question 9

Can F be the empty set?

Question 9

Can F be the empty set?

Yes. A DFA without any accepting states rejects every string

Question 10

Can F be all of Q?

Question 10

Can F be all of Q?

Yes. A DFA where every state is an accepting state accepts every string

Question 11

Can M have multiple start states?

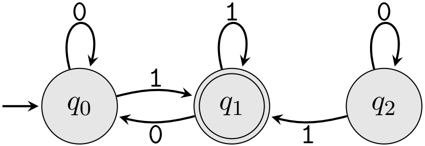
Question 11

Can M have multiple start states?

No. DFAs have a single start state

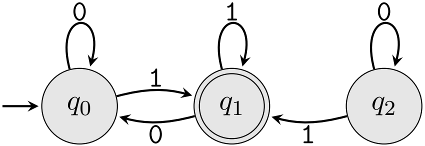
Question 12

Can a DFA have a state that's not reachable from any other state?



Question 12

Can a DFA have a state that's not reachable from any other state?



Yes. Nothing in the mathematical definition of a DFA forbids that and it simplifies conversions to DFA from other machines

Question 13

Can a DFA have a state without any transitions from it?

Question 13

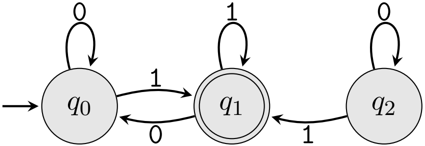
Can a DFA have a state without any transitions from it?
No. The transition function $\delta : Q \times \Sigma \to Q$ requires every state have a transition for every symbol in the alphabet

Recognition and acceptance

- A DFA accepts a string when the sequence of states it goes through when it runs on the string ends in an accepting state
- A DFA recognizes a language when it accepts every string in the language and, crucially, *rejects every string not in the language*

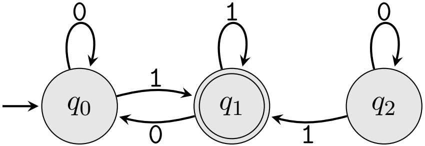
Question 14

Does this DFA recognize the string `1101`?



Question 14

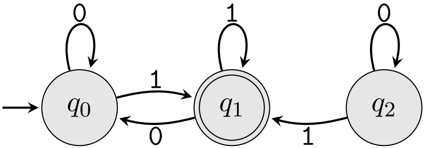
Does this DFA recognize the string 1101?



No. The question doesn't even make sense. DFAs recognize languages, not strings

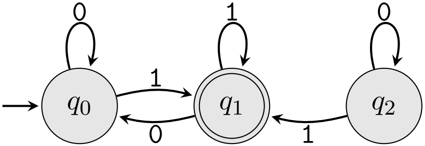
Question 15

Consider the language $A = \{w \mid w \in \{0,1\}^* \text{ ends in } 11\}$. The following DFA accepts every string in A. Does the DFA recognize A?



Question 15

Consider the language $A = \{w \mid w \in \{0,1\}^* \text{ ends in } 11\}$. The following DFA accepts every string in A. Does the DFA recognize A?



No. The DFA accepts string 1 which is not in A

Two methods of proving that a DFA recognizes a language

If we want to show that DFA M recognizes some language L, we have two options

1. Show that M accepts every string in L and rejects every string not in L
2. Show that M accepts every string in L and every string accepted by the DFA is in L

Nondeterministic finite automata

NFAs are five-tuples $N = (Q, \Sigma, \delta, q_0, F)$ where

- Q is the set of states
- Σ is the alphabet
- δ is the transition function
- q_0 is the start state
- F is the set of accepting states

Question 16

NFAs add two capabilities to DFAs

1. The ability to transition on an input symbol to zero or more states
2. The ability to transition on no input at all (ε-transitions)

For an NFA $N = (Q, \Sigma, \delta, q_0, F)$, is $\varepsilon \in \Sigma$?

Question 16

NFAs add two capabilities to DFAs

1. The ability to transition on an input symbol to zero or more states
2. The ability to transition on no input at all (ε-transitions)

For an NFA $N = (Q, \Sigma, \delta, q_0, F)$, is $\varepsilon \in \Sigma$?

No. Remember, the transition function is $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Question 17

Can an NFA have multiple start states?

Question 17

Can an NFA have multiple start states?

No. Still just the one

Question 18

Consider a new type of finite automaton called a multinondeterministic finite automaton (I just made this name up) which is a five tuple $M = (Q, \Sigma, \delta, I, F)$ where I is a set of initial states but is otherwise similar to an NFA.

Are MNFAs more powerful (meaning, can the class of MNFAs recognize more languages) than NFAs?

Question 18

Consider a new type of finite automaton called a multinondeterministic finite automaton (I just made this name up) which is a five tuple $M = (Q, \Sigma, \delta, I, F)$ where I is a set of initial states but is otherwise similar to an NFA.

Are MNFAs more powerful (meaning, can the class of MNFAs recognize more languages) than NFAs?

No. We can build an equivalent NFA by adding a new state which is the only start state and adding ε-transitions to the states in I.

Regular expressions

Regular expressions are defined recursively with three base cases

- $\underline{\varnothing}$ generates the empty language $\varnothing$
- $\underline{\varepsilon}$ generates the language $\{\varepsilon\}$
- $\underline{t}$ for some $t \in \Sigma$ generates the language $\{t\}$

and three recursive cases

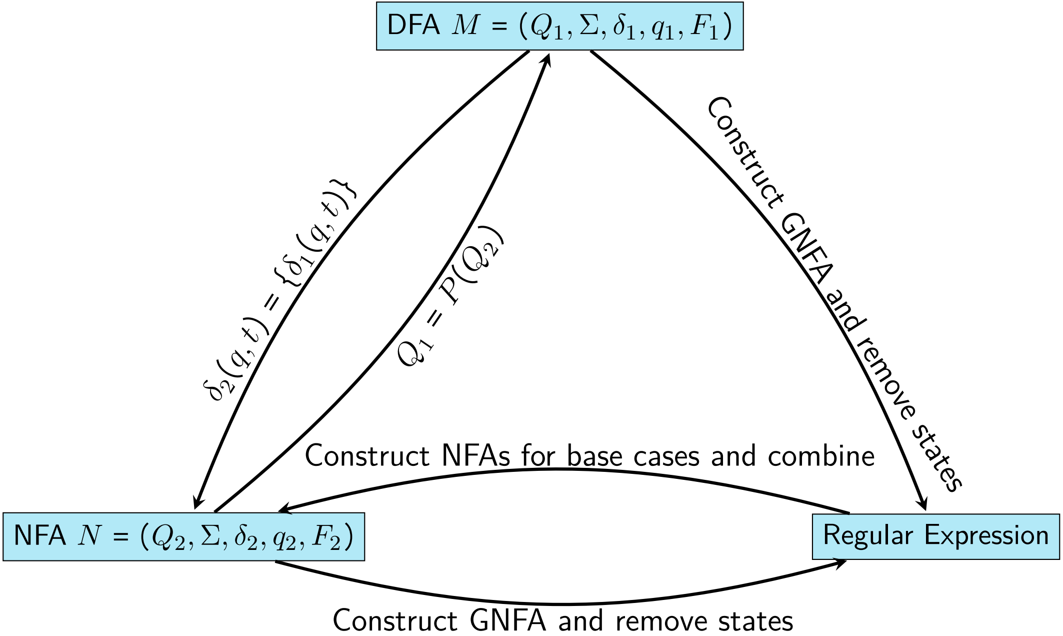
- $\underline{\mathtt{R}_1\mathtt{R}_2}$ generates $L(R_1) \circ L(R_2)$
- $\underline{\mathtt{R}_1 \mid \mathtt{R}_2}$ generates $L(R_1) \cup L(R_2)$
- $\underline{\mathtt{R}^*}$ generates $L(R)^*$

Regularity

Four equivalent statements about a language A

1. A is regular
2. Some DFA recognizes A
3. Some NFA recognizes A
4. Some regular expression generates A

Converting between DFA, NFA, regex



Converting from a regular expression to an NFA

Construct it step by step

1. Start with the base cases
2. Then construct NFAs for increasingly larger expressions by combining NFAs for smaller expressions

Example

Construct an NFA corresponding to the regular expression $\underline{(\texttt{aba} \mid \texttt{aa})^*}$

Converting from an NFA to a DFA

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we can construct an equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$

1. Each state of M represents a set of states of N
2. Each transition of M from state $S \subseteq Q$ on input t is to the state representing all of the states of N reachable from some state in S by following t and then 0 or more ε-transitions
3. The start state of M is the state that represents all of the states of N reachable from q_0 by following 0 or more ε-transitions
4. The set of accepting states of M are those representing a set of states of N that contains at least one accepting state of N

Formally,

1. $Q' = P(Q)$
2. $\delta'(S, t) = \bigcup_{q \in S} E\big(\delta(q, t)\big)$
3. $q_0' = E\big(\{q_0\}\big)$
4. $F' = \{S \mid S \subseteq Q \text{ and } S \cap F \neq \varnothing\}$

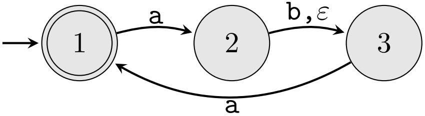
The function $E(\cdot)$ is the epsilon closure.

Example

Let's simplify our NFA for the language $\underline{(\mathtt{aba} \mid \mathtt{aa})^*}$.

Example

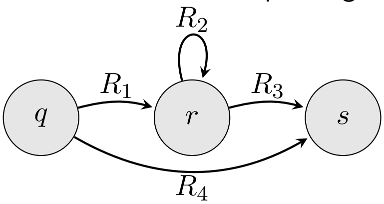
Let's simplify our NFA for the language $\underline{(\texttt{aba} \mid \texttt{aa})^*}$.

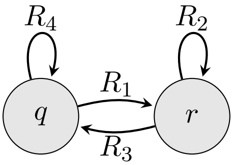


Now let's convert it to a DFA

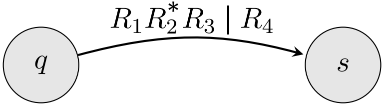
Converting from a DFA or an NFA to a regular expression

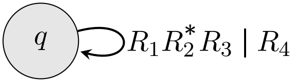
1. Create a GNFA by adding a start state and an accepting state
2. Add ε-transition from the new start state to the old start sate
3. Add ε-transitions from the old accepting states to the new accepting state
4. Convert each transition to a regex (i.e., transitions labeled `a`, `b` become `a | b`)
5. Remove each state, updating transitions from





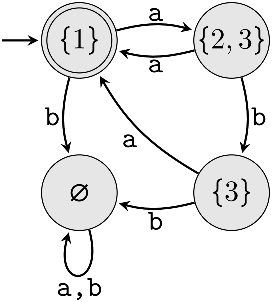
to





Example

Let's convert our DFA to a regular expression



Cartesian product construction

We can use DFAs directly to show that the class of regular languages is closed under union and intersection

Let

$$\begin{aligned} M_1 &= (Q_1, \Sigma, \delta_1, q_1, F_1) \\ M_2 &= (Q_2, \Sigma, \delta_2, q_2, F_2) \end{aligned}$$

and build

$$\begin{aligned} M &= (Q, \Sigma, \delta, q_0, F) \\ Q &= Q_1 \times Q_2 \\ \delta\big((q,r),t\big) &= \big(\delta_1(q,t),\ \delta_2(r,t)\big) \\ q_0 &= (q_1, q_2) \end{aligned}$$

For union, let $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
For intersection, let $F = F_1 \times F_2$

Pumping lemma

Theorem

Pumping lemma for regular languages For every regular language A, there exists an integer $p > 0$ called the pumping length such that for every $w \in A$ there exist strings x, y, and z with $w = xyz$ such that

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

A two-player game

Player One ($\exists$)		**Player Two ($\forall$)**
Claims A is pumpable with p.l. p		
	$\xrightarrow{A, p}$	
		Picks $w \in A$ s.t. $\lvert w \rvert \geq p$
	$\xleftarrow{w}$	
Picks x, y, z s.t. $w = xyz$		
	$\xrightarrow{x, y, z}$	
		Checks 3 conditions

Player One "wins" if

1. $xy^i z \in A$ for all $i \geq 0$
2. $\lvert y \rvert > 0$
3. $\lvert xy \rvert \leq p$

Can play as either Player One or Two

- To show that A is pumpable, play as Player One
  You must consider all possible w and pick x, y, and z
- To show that A is not pumpable, play as Player Two
  You must pick w and consider all possible x, y, and z

Proving that a language isn't regular

Three options

1. Assume that it is regular and show that it violates the pumping lemma
2. Assume that it is regular and apply operations on languages that preserve regularity, arrive at a contradiction because the result isn't regular
3. First apply some operations on languages, then use the pumping lemma

Closure properties of regular languages

The class of regular languages is closed under

- Union
- Concatenation
- Kleene star
- Intersection
- Complement
- Reversal
- Difference (we haven't proved this)
- Prefix
- Suffix (we haven't proved this)
- Left quotient by a string
- Right quotient by a string (we haven't proved this)
- Left/right quotient by a language (we haven't proved this)
- ...

Closure properties of nonregular languages

The class of nonregular languages is closed under

- Complement
- Reversal

The class of nonregular languages is *not* closed under

- Union
- Concatenation (we haven't proved this)
- Kleene star (we haven't proved this)
- Intersection
- Prefix (we haven't proved this)
- Suffix (we haven't proved this)
- Left/right quotient by a string/language (we haven't proved this)