A non-regular language

Example

$L = \{0^n 1^n : n \geq 0\}$ is not regular.

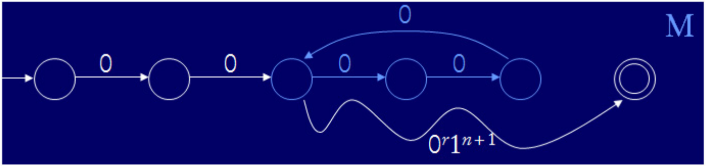
We reason by contradiction:

- Suppose we have managed to construct a DFA M for L.
- We argue something must be wrong with this DFA.
- In particular, M must accept some strings outside L.

A non-regular language

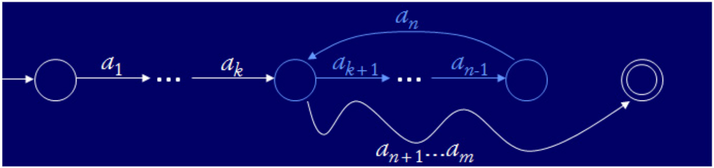
Imaginary DFA for L with n states. What happens when we run M on input $x = 0^{n+1}1^{n+1}$?

- M better accept, because $x \in L$.
- But since M has n states, it must revisit at least one of its states while reading 0^{n+1}.
- But then the DFA must contain a loop with 0s
- The DFA will then also accept strings that go around the loop multiple times
- But such strings have more 0s than 1s, so they are not in L!
- A contradiction!!!



General method for showing non-regularity

Every regular language L has a property:



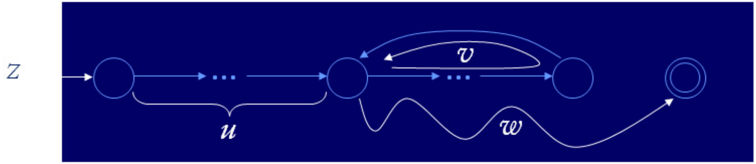
For every sufficiently long input z in L, there is a middle part in z that, even if repeated any number of times, keeps the input inside L

Pumping lemma: For every regular language L

Lemma

There exists a number n such that for every string z in L, we can write $z = u \cdot v \cdot w$ where

1. $|uv| \leq n$
2. $|v| \geq 1$
3. *For every $i \geq 0$, the string $uv^i w$ is in L.*



If L is regular, then:

There exists n such that for every z in L, we can write $z = u\,v\,w$ where ① $|uv| \leq n$, ② $|v| \geq 1$ and

(3) For every $i \geq 0$, the string uv^iw is in L.

So to prove L is not regular, it is enough to show:

proof strategy

For every n there exists z in L, such that for every way of writing $z = uvw$ where $|uv| \leq n$ and $|v| \geq 1$, the string uv^iw is not in L for some $i \geq 0$.

This is a game between you and an imagined adversary (say Donald)

	Donald	you
1	choose n	choose $z \in L$
2	write $z = uvw$ ($|uv| \leq n, |v| \geq 1$)	choose i
		you win if $uv^iw \notin L$

Arguing non-regularity

You need to give a strategy that, regardless of what the adversary does, always wins you the game.

	Donald	you
1	choose n	choose $z \in L$
2	write $z = uvw$ $(|uv| \leq n, |v| \geq 1)$	choose i
		you win if $uv^iw \notin L$

Example

	Donald	you
1	choose n	choose $z \in L$
2	write $z = uvw$ $(\lvert uv\rvert \leq n, \lvert v\rvert \geq 1)$	choose i
		you win if $uv^iw \notin L$

$L = \{0^n 1^n : n \geq 0\}$

	Donald	you
1	choose n	$z = 0^n 1^n$
2	write $z = uvw$	$i = 2$
	$\underbrace{00000000}_{u}\underbrace{0000}_{v}\underbrace{00011111111111111}_{w}$	$uv^2w = 0^{n+k}1^n \notin L$
	$\underbrace{00000000}_{u}\underbrace{0000}_{v}\underbrace{0000}_{v}\underbrace{00011111111111111}_{w}$	

Example

	Donald	you
1	choose n	choose $z \in L$
2	write $z = uvw$ $(\lvert uv \rvert \le n, \lvert v \rvert \ge 1)$	choose i
		you win if $uv^iw \notin L$

$L = \{1^p : p \text{ is prime }\}$

	Donald	you
1	choose n	
2	write $z = uvw = 1^a1^b1^c$	$i = a + c$

$$\underbrace{111111111}_{u=1^a}\underbrace{111}_{v=1^b}\underbrace{1111111111111111}_{w=1^c}$$

$$\begin{aligned} uv^iw &= 1^a1^{ib}1^c \\ &= 1^{(a+c)+ib} \\ &= 1^{(a+c)+(a+c)b} \\ &= 1^{(a+c)(b+1)} \\ &= 1^{\text{composite}} \notin L_5 \end{aligned}$$

Pumping Lemma is not a Necessary Condition

Example

We know $L = \{b^m c^m | m > 0\}$ is not regular. Let us consider $L' = a^+ L \cup (b + c)^*$. L' is not regular. If L' would be regular, then we can prove that L is regular (using the closure properties we will see next). However, the Pumping lemma does apply for L' with $n = 1$.

This shows the Pumping lemma is not a necessary condition for a language to be regular.

Use of closure properties to show non-regularity

- We can easily prove $L_1 = \{0^n 1^n | n > 0\}$ is not a regular language.
- $L_2 =$ the set of strings with an equal number of 0's and 1's isn't either, but that fact is trickier to prove.
- Regular languages are closed under $\cap$.
- If L_2 were regular, then $L_2 \cap L(0^* 1^*) = L_1$ would be, but it isn't.

Closure properties

Let L and M be regular. Then $L = L(R) = L(D)$ and $M = L(S) = L(F)$ for regular expressions R and S, and DFA D and F.
We have seen that RL are closed under the following operations:

- Union : $L \cup M = L(R + S)$ or $L \cup M = L(D \bigoplus F)$
- Complement : $\bar{L} = L(\bar{D})$
- Intersection : $L \cap M = \overline{\bar{L} \cup \bar{M}}$ or $L \cap M = L(D \times F)$
- Difference : $L - M = L \cap \bar{M}$
- Concatenation : $LM = L(RS)$
- Closure : $L^* = L(R^*)$
- Prefix : $Prefix(L) = \{x \mid \exists y \in \Sigma^*, xy \in L\}$ (Hint: in D, make final all states in a path from the start state to final state)
- quotient, morphism, inverse morphism, substitution, ...

Quotient

Definition

$L_1, L_2 \subseteq \Sigma^*$, $L_1/L_2 = \{x \in \Sigma^* \mid \exists y \in L_2, xy \in L_1\}$.

Note: $Pref(L) = L/\Sigma^*$.

Theorem

$L, R \subseteq \Sigma^*$. If R is regular, then R/L is also regular.

Proof Idea: $F' = \{q \in Q \mid \exists y \in L, \hat{\delta}(q, y) \in F\}$

Example

$L = \{a^{n^2} \mid n \geq 0\}$. $L/L = \{a^{n^2-m^2} \mid m, n \geq 0\} = a(aa)^* + (a^4)^*$.

Morphisms

$$h : \Sigma \to \Delta^*$$
$$h : \Sigma^* \to \Delta^* \qquad h(xy) = h(x)h(y), h(\epsilon) = \epsilon$$
$$h : 2^{\Sigma^*} \to 2^{\Delta^*} \qquad h(L) = \bigcup_{x \in L}\{h(x)\}$$

Example

$h(0) = ab, h(1) = ba, h(2) = \epsilon.$
$h(00212) = ababba;$
$h(\{0^n21^n | n \geq 0\}) = \{(ab)^n(ba)^n | n \geq 0\}$

Theorem

$h(K \cup L) = h(K) \cup (L);$
$h(K \cdot L) = h(K) \cdot h(L);$
$h(K^*) = h(K)^*.$

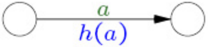
Inverse Morphisms

$h : \Sigma^* \to \Delta^*, K \subseteq \Delta^*$
$h^{-1}(K) = \{x \in \Sigma^* \mid h(x) \in K\}$

Theorem

Regular languages are closed under inverse morphism.

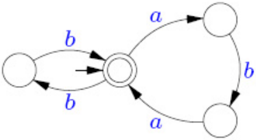
Proof Idea:

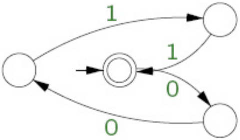


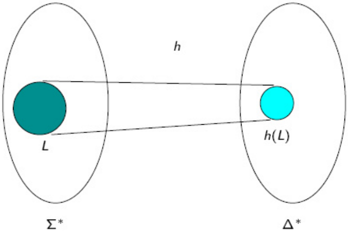
$$\delta'(p, a) = \delta(p, h(a))$$

$$h : 0 \mapsto ab, 1 \mapsto ba$$

$$h^{-1}(\ \{bb, aba\}^*\ ) = \{0011\}^*$$







Shuffle

Definition

$x\|\epsilon = \epsilon\|x = \{x\}$
$ax\|by = a(x\|by) \cup b(ax\|y)$
$K\|L = \bigcup_{x\in K, y\in L} x\|y$

$abb\|aca = \{aabbca, aabcba, aabcab, aacabb, aacbab,$
$aacbba, abbaca, ababca, abacba, abacab, acabba, acabab, acaabb\}.$

Theorem

If K, L are regular, so is $K\|L$.

Shuffle (cont'd)

Proof.

copies of alphabet

$$\Sigma,\ \Sigma_1 = \{\, a_1 \mid a \in \Sigma \,\},\ \Sigma_2 = \{\, a_2 \mid a \in \Sigma \,\}$$

$$h_1 : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto a \qquad a_2 \mapsto \epsilon$$

$$h_2 : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto \epsilon \qquad a_2 \mapsto a$$

$$g : \Sigma_1 \cup \Sigma_2 \to \Sigma^* \qquad a_1 \mapsto a \qquad a_2 \mapsto a$$

$$\begin{array}{ccccc} abbba & \xleftarrow{h_1} & a_1b_1a_2c_2b_1a_2c_2b_1a_1 & \xrightarrow{h_2} & acac \\ \in K & & \downarrow g & & \in L \\ & & abacbacba & & \end{array}$$

$$K \parallel L = g(\, h_1^{-1}(K) \cap h_2^{-1}(L)\,)$$

$\frac{1}{2}L$

Definition

$\frac{1}{2}L = \{x \in \Sigma^* | \exists y \in \Sigma^*, xy \in L; \quad |y| = |x|\}.$

Theorem

If L is regular, so is $\frac{1}{2}L$.

Proof.

guess middle state, simulate halves in paralle

$Q' = \{q_0'\} \cup Q \times Q \times Q$ (Note: middle, 1st, 2nd)

$\delta'(q_0', \epsilon) = \{[q, q_0, q] | q \in Q\}$ ϵ-move

$\delta'([q, p, r], a) = \{[q, \delta(p, a), \delta(r, b)] | b \in \Sigma\}$

$F' = \{[q, q, p] | q \in Q, p \in F\}$ $\square$

Decision Properties

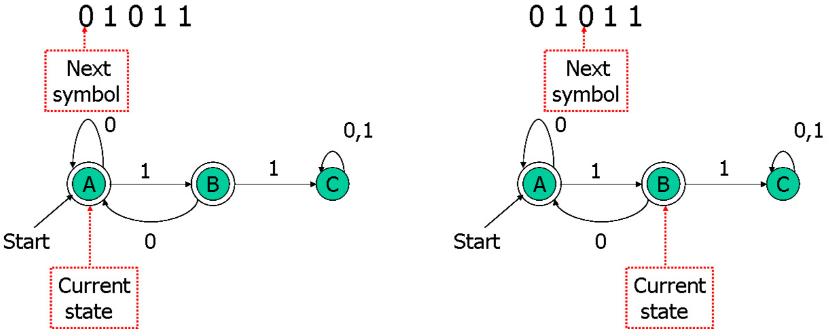
- A decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.
- Example: Is language L empty?
  - The representation is a DFA (or a RE that you will convert to a DFA).
  - Can you tell if $L(A) = \emptyset$ for DFA A?

Why Decision Properties

- When we talked about protocols represented as DFAs, we noted that important properties of a good protocol were related to the language of the DFA.
- Example: Does the protocol terminate? = Is the language finite?
- Example: Can the protocol fail? = Is the language nonempty?

The Membership Question

- Our first decision property is the question: is string w in regular language L?
- Assume L is represented by a DFA A.
- Simulate the action of A on the sequence of input symbols forming *w*



The Emptiness Problem

- Given a regular language, does the language contain any string at all.
- Assume representation is DFA.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes, else no.

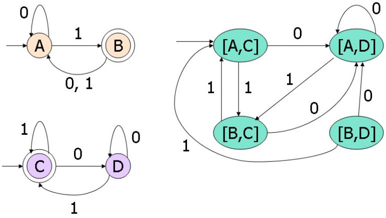
The Infiniteness Problem

- Is a given regular language infinite?
- Start with a DFA for the language.
- Key idea: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- Otherwise, the language is surely finite. Limited to strings of length n or less.
- There are an infinite number of strings of length $> n$, and we cant test them all.
- Second key idea: if there is a string of length $> n$ ($=$ number of states) in L, then there is a string of length between n and $2n - 1$.
- Test for membership all strings of length between n and $2n - 1$. If any are accepted, then infinite, else finite.

The Equivalence Problem

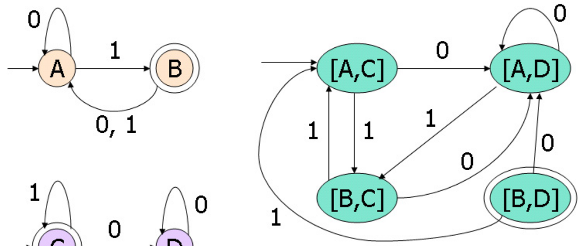
- Given regular languages L and M, is $L = M$?
- Algorithm involves constructing the product DFA from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states $Q \times R$. I.e., pairs $[q, r]$ with q in Q, r in R.

- Make the final states of the product DFA be those states $[q, r]$ such that exactly one of q and r is a final state of its own DFA. Thus, the product accepts w iff w is in exactly one of L and M.
- The product DFA's language is empty iff L = M.



The Containment Problem

- Given regular languages L and M, is $L \subseteq M$?
- Algorithm also uses the product automaton.
- How do you define the final states $[q, r]$ of the product so its language is empty iff $L \subseteq M$?
  - Answer: q is final; r is not.



Note: the only final state is unreachable, so containment holds.

The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA's we can, given a DFA A find the DFA with the fewest states accepting $L(A)$.
- Test all smaller DFA's for equivalence with A.
- But that's a terrible algorithm.

– **Efficient State Minimization**

- Construct a table with all pairs of states.
- If you find a string that distinguishes two states (takes exactly one to an accepting state), mark that pair.
- Algorithm is a recursion on the length of the shortest distinguishing string.

State Minimization

- Basis: Mark a pair if exactly one is a final state.
- Induction: mark $[q, r]$ if there is some input symbol a such that $[\delta(q, a), \delta(r, a)]$ is marked.
- After no more marks are possible, the unmarked pairs are equivalent and can be merged into one state.

Note: (Transitivity of Indistinguishable) If state p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.

Constructing the Minimum-State DFA

- Suppose $q_1, ..., q_k$ are indistinguishable states.
- Replace them by one state q.
- Then $\delta(q_1, a), ..., \delta(q_k, a)$ are all indistinguishable states.
  - Key point: otherwise, we should have marked at least one more pair.
- Let $\delta(q, a) =$ the representative state for that group.

Example: State Minimization

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}	{2,4,6,8}	{1,3,5,7,9}

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

Here it is with more convenient state names

Example: State Minimization

Start with marks for the pairs with one of the final states F or G.

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X				
B	X	X				
C	X	X				
D	X	X				
E	X	X				
F						

Example: State Minimization

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	x	x				
B	x	x				
C	x	x				
D	x	x				
E	x	x				
F						

Input r gives no help, because the pair [B, D] is not marked.

Example: State Minimization

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X	X	X	X	
B	X	X	X	X	X	
C	X	X				
D	X	X				
E	X	X				
F	X					

But input b distinguishes {A,B,F} from {C,D,E,G}. For example, [A, C] gets marked because [C, F] is marked.

Example: State Minimization

	r	b
→ A	B	C
B	D	E
C	D	F
D	D	G
E	D	G
* F	D	C
* G	D	G

	G	F	E	D	C	B
A	X	X	X	X	X	
B	X	X	X	X	X	
C	X	X	X	X		
D	X	X				
E	X	X				
F	X					

[C, D] and [C, E] are marked because of transitions on b to marked pair [F, G].

Example: State Minimization

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
∗	F	D	C
∗	G	D	G

	G	F	E	D	C	B
A	X	X	X	X	X	X
B	X	X	X	X	X	
C	X	X	X	X		
D	X	X				
E	X	X				
F	X					

[A, B] is marked because of transitions on r to marked pair [B, D].

[D, E] can never be marked, because on both inputs they go to the same state.

Example: State Minimization

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

		r	b
→	A	B	C
	B	H	H
	C	H	F
	H	H	G
*	F	H	C
*	G	H	G

	G	F	E	D	C	B
A	X	X	X	X	X	X
B	X	X	X	X	X	
C	X	X	X	X		
D	X	X				
E	X	X				
F	X					

Replace D and E by H.
Result is the minimum-state DFA.