Languages That Are and Are Not Regular

Read L & S 2.5, 2.6
Read Supplementary Materials: Regular Languages and Finite State Machines: The Pumping Lemma for Regular Languages.
Do Homework 9.

Deciding Whether a Language is Regular

Theorem: There exist languages that are not regular.
Lemma: There are an uncountable number of languages.
Proof of Lemma:
Let: Σ be a finite, nonempty alphabet, e.g., {a, b, c}.

Then Σ^* contains all finite strings over Σ.
e.g., {ε, a, b, c, aa, ab, bc, abc, bba, bbaa, bbbaac}

Σ^* is countably infinite, because its elements can be enumerated one at a time, shortest first.

Any language L over Σ is a subset of Σ^*, e.g., L1 = {a, aa, aaa, aaaa, aaaaa, …}
L2 = {ab, abb, abbb, abbbb, abbbbb, …}
The set of all possible languages is thus the power set of Σ^*.

The power set of any countably infinite set is not countable. So there are an uncountable number of languages over Σ^*.

Some Languages Are Not Regular

Theorem: There exist languages that are not regular.
Proof:
(1) There are a countably infinite number of regular languages. This true because every description of a regular language is of finite length, so there is a countably infinite number of such descriptions.
(2) There are an uncountable number of languages.

Thus there are more languages than there are regular languages. So there must exist some language that is not regular.

Showing That a Language is Regular

Techniques for showing that a language L is regular:
1. Show that L has a finite number of elements.
2. Exhibit a regular expression for L.
3. Exhibit a FSA for L.
4. Exhibit a regular grammar for L.
5. Describe L as a function of one or more other regular languages and the operators $\cdot, \cup, \cap, *, -, \neg$. We use here the fact that the regular languages are closed under all these operations.
6. Define additional operators and prove that the regular languages are closed under them. Then use these operators as in 5.

Example

Let Σ = {0, 1, 2, … 9}
Let L $\subseteq \Sigma^*$ be the set of decimal representations for nonnegative integers (with no leading 0's) divisible by 2 or 3.

L_1 = decimal representations of nonnegative integers without leading 0's.
$L_1 = 0 \cup \{1, 2, \ldots 9\}\{0 - 9\}^*$
So L_1 is regular.

L_2 = decimal representations of nonnegative integers without leading 0's divisible by 2
$L_2 = L_1 \cap \Sigma^*\{0, 2, 4, 6, 8\}$
So L_2 is regular.

Example, Continued

$L_3 = L_1$ and divisible by 3

Recall that a number is divisible by 3 if and only if the sum of its digits is divisible by 3. We can build a FSM to determine that and accept the language L_{3a}, which is composed of strings of digits that sum to a multiple of 3.

$L_3 = L_1 \cap L_{3a}$

Finally, $L = L_2 \cup L_3$

Another Example

$\Sigma = \{0 - 9\}$
L = {w : w is the social security number of a living US resident}

Finiteness - Theoretical vs. Practical

Any finite language is regular. The size of the language doesn't matter.

Parity ⟵ ⟶ Soc. Sec. #

Checking Checking

But, from an implementation point of view, it very well may.

When is an FSA a good way to encode the facts about a language?

What are our alternatives?

FSA's are good at looking for repeating patterns. They don't bring much to the table when the language is just a set of unrelated strings.

Showing that a Language is Not Regular

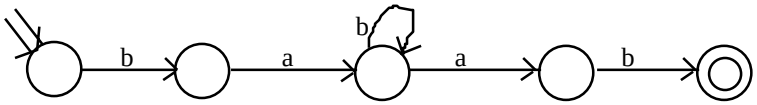
The argument, "I can't find a regular expression or a FSM", won't fly. (But a proof that there cannot exist a FSM is ok.)

Instead, we need to use two fundamental properties shared by regular languages:

1. We can only use a finite amount of memory to record essential properties.
   Example:
   $a^n b^n$ is not regular

2. The only way to generate/accept an infinite language with a finite description is to use Kleene star (in regular expressions) or cycles (in automata). This forces some kind of simple repetitive cycle within the strings.
   Example:
   ab*a generates aba, abba, abbba, abbbba, etc.
   Example:
   $\{a^n : n \geq 1$ is a prime number$\}$ is not regular.

Exploiting the Repetitive Property



If a FSM of n states accepts any string of length ≥ n, how many strings does it accept?

L = bab*ab

n

b a b b b b a b

x y z

xy*z must be in L.

So L includes: baab, babab, babbab, babbbbbbbbbbab

The Pumping Lemma for Regular Languages

If L is regular, then
- ∃ N ≥ 1, such that
  - ∀ strings w ∈ L, where |w| ≥ N,
    - ∃ x, y, z, such that w = xyz
      - and |xy| ≤ N,
      - and y ≠ ε,
      - and ∀ q ≥ 0, xy^qz is in L.

Example: L = a^nb^n

a a a a a a a a a b b b b b b b b b b

x y z

∃ N ≥ 1	Call it N
∀ long strings w	We pick one
∃ x, y, z	We show no x, y, z

Example: a^nb^n is not Regular

N is the number from the pumping lemma (or one more, if N is odd).

Choose w = $a^{N/2}b^{N/2}$. (Since this is what it takes to be "long enough": |w| ≥ N)

1 2

a a a a a a a a a a | b b b b b b b b b b

x y z

We show that there is no x, y, z with the required properties:

|xy| ≤ N,
y ≠ ε,
∀ q ≥ 0, xy^qz is in L.

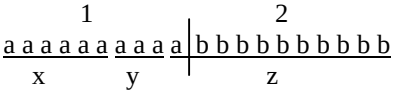
Three cases to consider:

- y falls in region 1:
- y falls across regions 1 and 2:
- y falls in region 3:

Example: $a^n b^n$ is not Regular

Second try:

Choose w to be be $a^N b^N$. (Since we get to choose any w in L.)



We show that there is no x, y, z with the required properties:

$|xy| \le N$,
$y \ne \varepsilon$,
$\forall\ q \ge 0$, $xy^q z$ is in L.

Since $|xy| \le N$, y must be in region 1. So $y = a^g$ for some $g \ge 1$. Pumping in or out (any q but 1) will violate the constraint that the number of a's has to equal the number of b's.

A Complete Proof Using the Pumping Lemma

Proof that $L = \{a^n b^n\}$ is not regular:

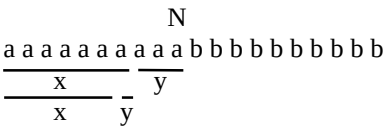
Suppose L is regular. Since L is regular, we can apply the pumping lemma to L. Let N be the number from the pumping lemma for L. Choose $w = a^N b^N$. Note that $w \in L$ and $|w| \ge N$. From the pumping lemma, there exists some x, y, z where xyz = w and $|xy| \le N$, $y \ne \varepsilon$, and $\forall\ q \ge 0$, $xy^q z \in L$. Because $|xy| \le N$, $y = a^{|y|}$ (y is all a's). We choose q = 2 and $xy^q z = a^{N+|y|} b^N$. Because $|y| > 0$, then $xy^2 z \notin L$ (the string has more a's than b's). Thus for all possible x, y, z: xyz = w, $\exists q$, $xy^q z \notin L$. Contradiction. $\therefore$ L is not regular.

Note: the underlined parts of the above proof is "boilerplate" that can be reused. A complete proof should have this text or something equivalent.

You get to choose w. Make it a single string that depends only on N. Choose w so that it makes your proof easier.
You may end up with various cases with different q values that reach a contradiction. You have to show that all possible cases lead to a contradiction.

Proof of the Pumping Lemma

Since L is regular it is accepted by some DFSA, M. Let N be the number of states in M. Let w be a string in L of length N or more.



Then, in the first N steps of the computation of M on w, M must visit N+1 states. But there are only N different states, so it must have visited the same state more than once. Thus it must have looped at least once. We'll call the portion of w that corresponds to the loop y. But if it can loop once, it can loop an infinite number of times. Thus:

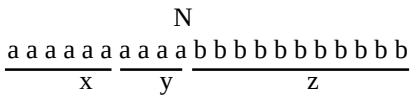
- M can recognize $xy^q z$ for all values of $q \ge 0$.
- $y \ne \varepsilon$ (since there was a loop of length at least one)
- $|xy| \le N$ (since we found y within the first N steps of the computation)

.

Another Pumping Example

$L = \{w=a^Jb^K : K > J\}$ (more b's than a's)

Choose $w = a^Nb^{N+1}$

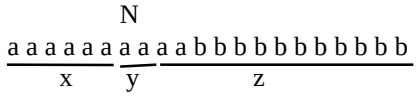


We are guaranteed to pump only a's, since $|xy| \leq N$. So there exists a number of copies of y that will cause there to be more a's than b's, thus violating the claim that the pumped string is in L.

A Slightly Different Example of Pumping

$L = \{w=a^Jb^K : J > K\}$ (more a's than b's)

Choose $w = a^{N+1}b^N$



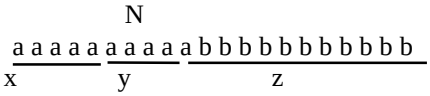
We are guaranteed that y is a string of at least one a, since $|xy| \leq N$. But if we pump in a's we get even more a's than b's, resulting in strings that are in L.

What can we do?

Another Slightly Different Example of Pumping

$L = \{w=a^Jb^K : J \geq K\}$

Choose $w = a^{N+1}b^N$



We are guaranteed that y is a string of at least one a, since $|xy| \leq N$. But if we pump in a's we get even more a's than b's, resulting in strings that are in L.

If we pump out, then if y is just a then we still have a string in L.

What can we do?

Another Pumping Example

$L = aba^nb^n$

Choose $w = aba^Nb^N$

$$\underbrace{a\ b}_{x}\ \underbrace{a}_{y}\ \underbrace{a\ a\ a\ a\ a\ a\ a\ a\ b\ b\ b\ b\ b\ b\ b\ b\ b}_{z} \qquad (N)$$

What are the choices for (x, y):
(ε, a)
(ε, ab)
(ε, aba^+)
(a, b)
(a, ba^+)
(aba*, a^+)

What if L is Regular?

Given a language L that is regular, pumping will work: L = (ab)* Choose $w = (ab)^N$

There must exist an x, y, and z where y is pumpable.

$$\underbrace{abababab}_{x}\ \underbrace{ababab}_{y}\ \underbrace{abababababab}_{z}$$

Suppose y = ababab Then, for all $q \geq 0$, $x\,y^q z \in L$

Note that this does not prove that L is regular. It just fails to prove that it is not.

Using Closure Properties

Once we have some languages that we can prove are not regular, such as a^nb^n, we can use the closure properties of regular languages to show that other languages are also not regular.

Example: $\Sigma = \{a, b\}$
$L = \{w : w$ contains an equal number of a's and b's $\}$
a*b* is regular. So, if L is regular, then $L_1 = L \cap a^*b^*$ is regular.

But L_1 is precisely a^nb^n. So L is not regular.

Don't Try to Use Closure Backwards

One Closure Theorem:
If L_1 and L_2 are regular, then so is $L_3 = L_1 \cap L_2$.

But what if L_3 and L_1 are regular? What can we say about L_2?

$L_3 = L_1 \cap L_2$.

Example: $ab = ab \cap a^nb^n$

A Harder Example of Pumping

$\Sigma = \{a\}$
$L = \{w = a^K : K \text{ is a prime number}\}$

N
a a a a a a a a a a a a a
x y z

$|x| + |z|$ is prime.
$|x| + |y| + |z|$ is prime.
$|x| + 2|y| + |z|$ is prime.
$|x| + 3|y| + |z|$ is prime, and so forth.

Distribution of primes:

Distribution of $|x| + q|y| + |z|$:

But the Prime Number Theorem tells us that the primes "spread out", i.e., that the number of primes not exceeding x is asymptotic to x/ln x.

Note that when $q = |x| + |z|$, $|xy^q z| = (|y| + 1) \times (|x| + |z|)$, which is composite (non-prime) if both factors are > 1. If you're careful about how you choose N in a pumping lemma proof, you can make this true for both factors.

Automata Theory is Just the Scaffolding

Our results so far give us tools to:

- Show a language is regular by:
  - Showing that it has a finite number of elements,
  - Providing a regular expression that defines it,
  - Constructing a FSA that accepts it, or
  - Exploiting closure properties
- Show a language is not regular by:
  - Using the pumping lemma, or
  - Exploiting closure properties.

But to use these tools effectively, we may also need domain knowledge (e.g., the Prime Number Theorem).

More Examples

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7\}$
$L = \{w = \text{the octal representation of a number that is divisible by 7}\}$

Example elements of L:
7, 16 (14), 43 (35), 61 (49), 223 (147)

More Examples

$\Sigma = \{W, H, Q, E, S, T, B \text{ (measure bar)}\}$
$L = \{w = w \text{ represents a song written in 4/4 time}\}$

Example element of L:
WBWBHHBHQQBHHBQEEQEEB

More Examples

$\Sigma = \{0 - 9\}$
L = {w = is a prime Fermat number}

The Fermat numbers are defined by

$$F_n = 2^{2^n} + 1, n = 1, 2, 3, \ldots$$

Example elements of L:
$F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65{,}537$

Another Example

$\Sigma = \{0 - 9, *, =\}$
L = {w = a*b=c: a, b, c $\in$ $\{0\text{-}9\}^+$ and int(a) * int(b) = int(c)}

The Bottom Line

A language is regular if:

OR

The Bottom Line (Examples)

- The set of decimal representations for nonnegative integers divisible by 2 or 3
- The social security numbers of living US residents.
- Parity checking
- $a^n b^n$
- $a^j b^k$ where k>j
- a^k where k is prime
- The set of strings over {a, b} that contain an equal number of a's and b's.
- The octal representations of numbers that are divisible by 7
- The songs in 4/4 time
- The set of prime Fermat numbers

Decision Procedures

A **decision procedure** is an algorithm that answers a question (usually "yes" or "no") and terminates. The whole idea of a decision procedure itself raises a new class of questions. In particular, we can now ask,

1. Is there a decision procedure for question X?
2. What is that procedure?
3. How efficient is the best such procedure?

Clearly, if we jump immediately to an answer to question 2, we have our answer to question 1. But sometimes it makes sense to answer question 1 first. For one thing, it tells us whether to bother looking for answers to questions 2 and 3.

Examples of Question 1:

Is there a decision procedure, given a regular expression E and a string S, for determining whether S is in L(E)?

Is there a decision procedure, given a Turing machine T and an input string S, for determining whether T halts on S?

Decision Procedures for Regular Languages

Let M be a deterministic FSA. There is a decision procedure to determine whether:

- $w \in L(M)$ for some fixed w
- L(M) is empty
- L(M) is finite
- L(M) is infinite

Let M_1 and M_2 be two deterministic FSAs. There is a decision procedure to determine whether M_1 and M_2 are equivalent. Let L_1 and L_2 be the languages accepted by M_1 and M_2. Then the language

$L = (L_1 \cap \neg L_2) \cup (\neg L_1 \cap L_2)$

$= (L_1 - L_2) \cup (L_2 - L_1)$

must be regular. L is empty iff $L_1 = L_2$. There is a decision procedure to determine whether L is empty and thus whether $L_1 = L_2$ and thus whether M_1 and M_2 are equivalent.