Non-Regular Languages

As we have asked while studying the regular languages, a legitimate question is: *"Are all languages (i.e. sets of strings) regular*?" In fact, we have already discussed a language which was not regular, namely $a^n b^n$.

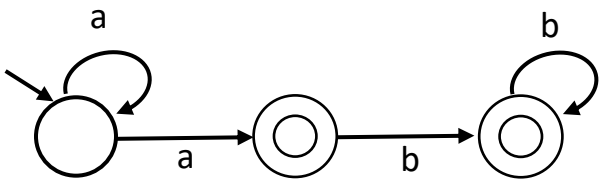
Given what we have just studied there are several other ways to ask the same question:

Are all languages recognized by	?
Are all languages represented by	?
Are all languages generated by	?

Let us look at two languages on {a, b} which appear similar but one is regular and the other one is not:

$L = \{ a^n b^m \mid n, m > 0\}$	which is regular
$L = \{ a^n b^n \mid n > 0\}$	which is not

Notice the difference? In the second case you need to count and "remember" the number of a's so that we count the same number of b's. In the first case we only need to make sure that all the a's come before the b's but there is no relationship between the number of a's and the number of b's so that we don't need to count. In other words, the first language exhibits a "simpler" pattern than the second one. If you think about it, it is not difficult to imagine an NFA for the first language:



However, for the second language we need to count (possibly to infinity), therefore no machine with a finite number of state will do so. In general, we need some theory which shows that "not all languages are regular" and allows us to distinguish between the languages which are regular and those which are not.

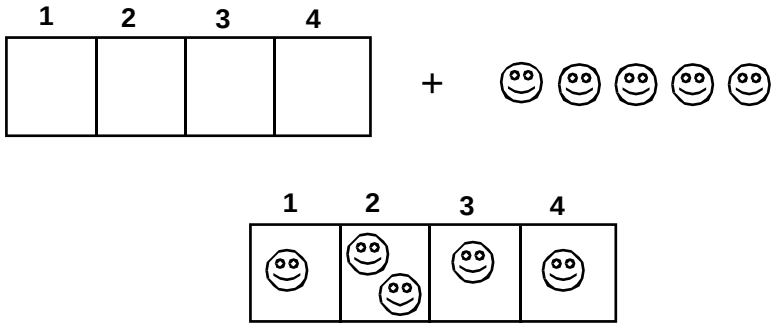
We will identify a characteristic unique to all regular languages (i.e. the regular languages possess that property while languages that are not regular do not). We will do so via the so called "pumping lemma."

Note that we only need to focus on infinite languages since we already know that all finite languages are regular (remember they can be described by a regular expression consisting of all the strings separated by '+').

Before we proceed let us discuss an important but simple concept.

The Pigeonhole Principle

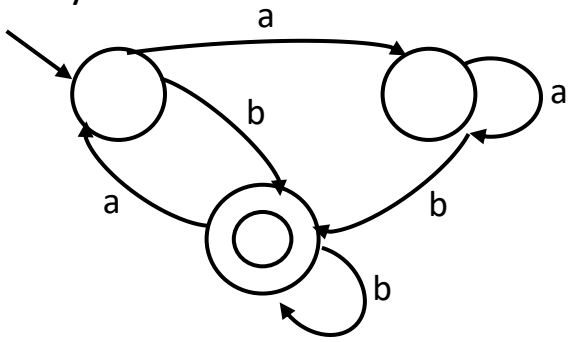
It is a big name for a simple idea. Imagine that we have to put n objects (the pigeons) into m boxes (the pigeonholes). If n > m there must be at least one box with more than one object in it (i.e. at least two pigeons in one pigeonhole).



The Pumping Lemma

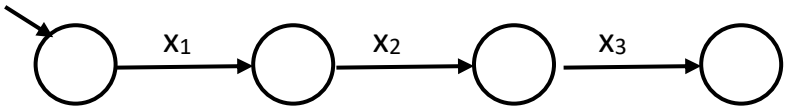
If a language L is regular, there exists a finite automaton (FA) M which recognizes exactly L (i.e., it accepts all the strings in L ***and no other***).

By definition such an FA has a finite number of states. Let us examine a very simple case where the FA has only three states:



What is the language accepted by this machine? (Note: this is not necessarily the simplest machine accepting that language).

Claim: any string of length 3 or more will take a path through the machine above where at least one state will be repeated. This is due to the fact that a path taken with a string of length 3 will go through 4 states:



This can be generalized: if a string is of length n its path must go through n + 1 states, not necessarily distinct.

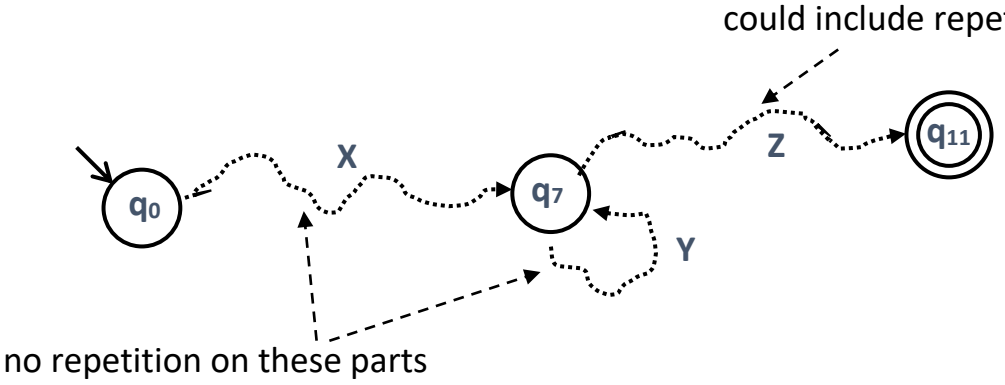
Let us now look at a larger automaton we will call M. For the sake of argument let us now peg the number of states M has to 15 (there is nothing special about 15 other than it is a finite number). If the language accepted by M is infinite , it must contain some strings longer than 15. Let us call "s" such a string and assume that it is of length n > 15. If we feed it to M, M will move through a sequence of states from the initial state to a final state. As pointed out earlier, it will move through n+1 states > 16, hence it should be clear that, since M has only 15 states, M will go through some of the states more than once. This is an application of the "pigeonhole principle" we saw earlier.

Let us, arbitrarily, assume that the sequence of states on the path taken by s through the machine M works as follows:

the path starts in $\mathbf{q_0}$ since it is the initial state, goes to $\mathbf{q_5}$, then to $\mathbf{q_{14}}$, then to $\mathbf{q_8}$, then to $\mathbf{q_7}$, etc., and that it finally lands in $\mathbf{q_{11}}$ which must be a final state since "s" is in the language accepted by M. Moreover, some state will be repeated since the string is longer than 15 and M has only 15 states. Let us assume that q_4 is a repeated state (we will assume that it corresponds to the first repetition of a state (careful that is not necessarily the first state we encountered that is repeated), hence the states which precede $\mathbf{q_7}$ [$\mathbf{q_0}$, $\mathbf{q_5}$, $\mathbf{q_{14}}$, $\mathbf{q_8}$] may be states which are repeated but the second instance of such state will happen after that of $\mathbf{q_4}$). Let us look at the path taken by "s" through M (remember states have been chosen arbitrarily):

$$
\begin{array}{lccccccccccc}
s = & s_1 & s_2 & s_3 & s_4 & s_5 & \ldots & s_i & \ldots & s_i & \ldots & s_n \\
 & | & | & | & | & | & \ldots & | & \ldots & | & & | \\
 & q_0 & q_5 & q_{14} & q_8 & \mathbf{q_7} & q_{12} & \ldots & \mathbf{q_7} & \ldots & q_5 & & q_{11}
\end{array}
$$

If we call "X" the portion of the string taking us from q_0 to the first occurrence of q_7, "Y" that portion of the string taking us from q_7 back to the next instance of q_7, and "Z" the end portion of the string that takes us from q_7 to the accepting state q_{11}, it corresponds to the following view of the path through the automaton:

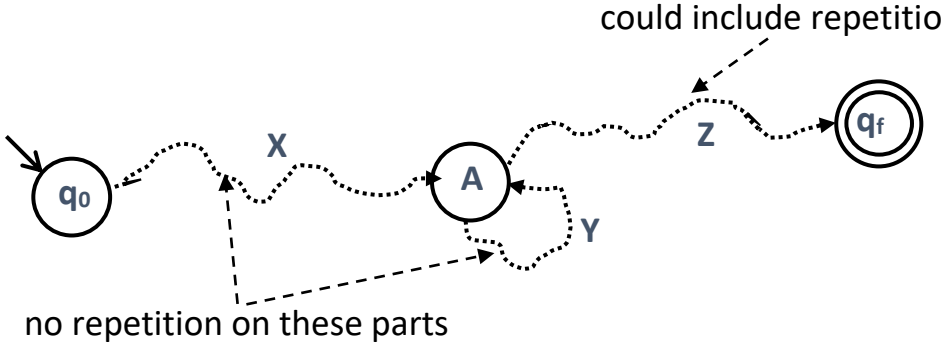


Clearly, if we remove Y the resulting string XZ is also accepted (the path will still go from q_0 to q_{11}). Similarly, if we add y two more times, giving the string XY^3Z, we will continue to have a string which is accepted. In general, all the strings of the form:

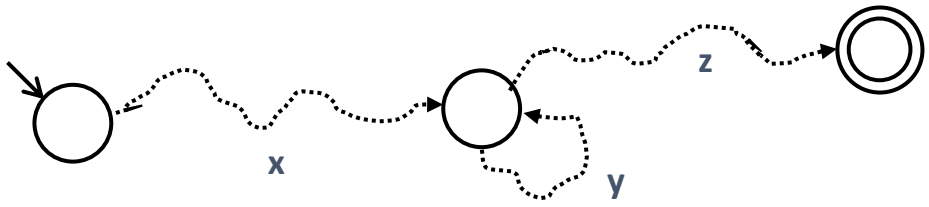
$$X\ Y^k\ Z, \qquad k \geq 0$$

will be accepted and therefore are in the language accepted by M.

Let us now look at the general case and assume that the automaton has **n states** and that the string is at least n long (i.e., its length is ≥ n). That means that the path will go through at least n+1 states. We get a picture similar to the one above:



where A is the first repeated state on the path, s = X Y Z, and Y represents that portion of the string which takes us from state A back to state A for the first time. **Note that, by definition, there cannot be any repeated state on the path X takes us through, or the path Y takes us through.**



Again .....do you see that:

- it means that xz, xyyz, xyyyz will also be accepted by the FA. In fact, do you see that all strings of the form x y* z will be accepted.
- since the language is infinite we can guarantee that there is such a string, hence $|y| > 0$.
- since A is the first repeated state xy has no repeated state and, since the machine has only n states we must have $|xy| \leq n$ (otherwise there would be another repeated state on the xy path).

It turns out this is a property of regular languages which is not shared by any other class of languages. It is expressed in the pumping lemma.

Note that the pumping lemma is not a sufficient condition (i.e. we can't use it to prove a language is regular as some non-regular languages may have those properties – but not all of them will). The pumping lemma is a necessary condition and we can use it to prove that a language is not regular (by proving it does not have the property) which is how the pumping lemma is used. This is done via a *proof by contradiction* (you assume the opposite of what you want to prove, namely you assume the language is regular, and show that this leads to a contradiction).

The pumping lemma states the following:

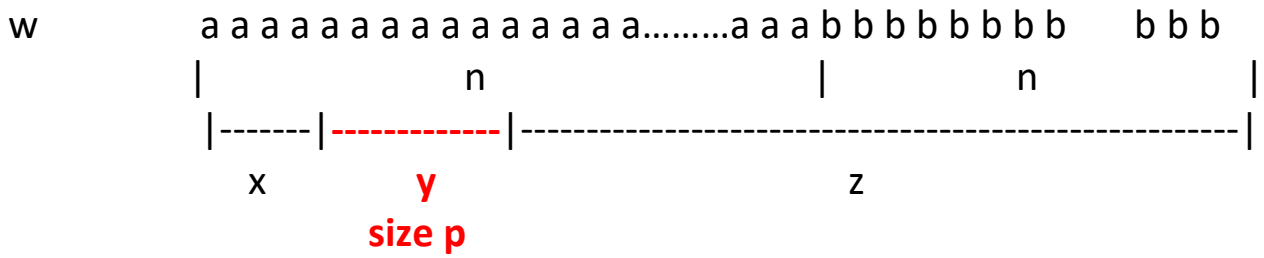
> If A is a regular language, then there is a number p (called the pumping length) such that, if s is any string in A of length at least p, s may be divided into three pieces x, y, and z, (i.e. s = xyz), such that all of the following hold:
> 1. for each $i \leq 0$, $xy^i z$ is in A
> 2. $|y| > 0$
> 3. $|xy| \leq p$
>
> *Condition 1: allows us to "pump in or out" elements in A. Note that **x** and **z** could be λ, but **y** cannot (see condition 2).*
> *Condition 2: without it the lemma is trivially true with **y** = λ.*
> *Condition 3: assures us we can pick **x** and **y** small, as needed.*

Let us return to our language $L = \{ a^k b^k \mid k \geq 0\}$.

1. Opposite of what we want to prove: **assume L is regular**.
2. Shows it leads to a contradiction:

> If L is regular, then the pumping lemma applies. Hence we know that, ***although we don't know its value***, there exists some integer $n \geq 0$ such that any *(long enough)* string **w** in L contains a **non-empty substring y** that can be "pumped" in, or "cut out" of, the word w. If this leads us to a string which, according to the pumping lemma should be in L, but in fact is not part of the language we will have proven that L is not regular.

In the case of $L = (a^k b^k)$, assume n is the pumping length. We need to cleverly choose a string *longer than n* and such that if we pump y, or cut it out, the resulting string is clearly not in L.
We will choose a string such that xy consists of a's only. The string $a^n b^n$ is such a string (it is longer than n and $|xy| \leq n$ guarantees that y contains only a's), furthermore call p the number of a's in y (i.e. $y = a^p$). In other words we have a picture like this:



From the pumping lemma we know that, if L is regular, removing y results in a string still in L. In other word $a^{n-p}b^n$ must be in L. Since p > 0 the number of a's no longer matches the number of b's. Hence $a^{n-p}b^n$ cannot be in L which contradicts the assumption that L is regular.

3. Conclusion: **L is not regular**

Your turn:

1. Palindromes are strings which can be read the same forward and backward. (example: Laval, atoyota, abccba, 0110110). Show that the language of palindrome is not regular.

   Let L = { w | w = w^R, w ∈ { a, b} } be the language of palindromes (w^R denotes the reverse of w).

2. Show that the language $L = \{a^k v \mid v \in \{a, b\}^* \ \& \ |v| = k\}$ is not regular.