The Pumping Lemma for Regular Languages

M. ASAD TARIQ

Note: This article assumes knowledge of finite automata and regular languages.

What if I tell you to design a finite automaton (whether deterministic or non-deterministic—doesn't matter which) for me which accepts the language $L = \{0^i 1^i \mid i \geq 0\}$? Go on, try to think how you would go about doing it. As it turns out, this is a language for which it simply isn't possible to make a finite automaton that accepts it. And it is not even a rare exception—there are loads of such languages. Now the question arises: how do we determine whether we can or cannot design a finite automaton corresponding to a particular language? After all, just because we are unable to do so doesn't mean it is actually impossible; perhaps, we're just not getting the right idea? Or, perhaps, our automata skills are just that bad? (Just joking! Even if you think you aren't too good at designing automata, it is something that you can easily get better at in a matter of time as long as you keep practising.)

This is where the *Pumping Lemma* comes in. (Note that there is another variant of the pumping lemma which applies to context-free languages. However, the one we are talking about here is the one for regular languages; so, whenever you see the word 'pumping lemma' in this article, take it as 'pumping lemma for regular languages'—as indicated in the title as well.) The pumping lemma describes a certain property that is held by all regular languages. Regular languages are those for which it is possible to design a finite automaton which accepts that language. Conversely, those languages for which this task is not possible (like the example given in the previous paragraph) are called irregular languages. So, if a language is regular, it has the property described by the pumping lemma.

Note that the converse is not necessarily true: even if a language does possess the property described by the lemma, that does mean it is a regular language for sure. There are languages that are not regular but still fulfil the lemma. But if a language is regular, then it must have the property. Now, this point is quite important. What this means is that we cannot use the pumping lemma to prove that a language is regular. Even if we show that a certain language possesses the property described by the lemma, we cannot claim for certain that it is regular. However, if we show that a given language does not possess the property described by the lemma, we can conclude with certainty that the language in question is not regular, and so it is not possible to design a finite automaton accepting it.

Let us now consider what exactly this property is which the pumping lemma claims is held by all regular languages. Consider a hypothetical finite automaton having n states. Now, make the following observation: if the automaton accepts a string having length greater than or equal to n, it must have visited at least one state more than once. Think about this for a moment. For each of the initial $n-1$ characters of the string, it is conceivable (though not necessary) that starting from the initial state, we visited a different state of the automaton (from the $n-1$ remaining states, not counting the initial state) each time, never repeating a state we've already visited. However, by the time we reach the n^{th} character of the string, all of the automaton's states have been visited before. As a result, for the string to get accepted, it has to visit at least one previously visited state again. (Note that it is possible that it visits several states multiple times, and does not visit some at all. The key point is just that it visits at least one state multiple times.)

What does this observation tell us? It tells us that the path (i.e. the sequence of states visited) followed by the string in question while getting accepted by the automaton contains a loop. That is, the path followed by the string went from state Q_0 to some state Q_k, then looped back to state Q_k (either directly, or via some intermediate states), and then went on to a final (i.e. accepting) state Q_n. Now, the interesting thing about it is, what if we did not traverse the loop at all? What if we went from Q_0 to Q_k, and then kept moving onwards to Q_n, without taking a detour at state Q_k? Naturally, the string would still get accepted, as it would still end up at an accepting state. This means that the part of the string corresponding to the transitions of the loop at Q_k could potentially be omitted, and the resulting string would for sure also be accepted by the automaton if the original string was accepted.

Not just that, but we could also do the opposite. Instead of just traversing the loop once, why not traverse it multiple times? That is, what we went from Q_0 to Q_k, looped back to Q_k, then again looped back to Q_k, then again, and again, and so on an arbitrary number of times, followed by moving on from Q_k to the final state Q_n? Once again, the string would naturally still get accepted. This means that we can potentially repeat the part of the string corresponding to the loop any number of times and all of the resulting strings would also get accepted like the original. Let us express this mathematically. Let's say we have a string s having $|s| \geq n$ that is accepted by a finite automaton. The condition about the string's length is necessary because the entire situation described above occurs only for strings having length greater than or equal to n. After all, if this condition does not hold, then it is not even necessary for there to be a loop in the string's path, is it?

Now, given our string s with $|s| \geq n$, there must be some possible way to divide it into three parts u, v and w such that u corresponds to the path before the loop (i.e. from Q_0 to Q_k as per our discussion above), v corresponds to the loop itself (i.e. from Q_k back to Q_k), and w corresponds to the path after the loop (i.e. from Q_k till Q_n). Given such a decomposition of s into three substrings, i.e. $s = uvw$, the string s' produced by either omitting the substring v, or repeating it multiple times, is also accepted by the automaton that accepts s. Mathematically, if $s \in L$, then $s' = uv^i w$ where $i = 0, 1, 2, \ldots$ is also part of L, i.e. $s' \in L$. (Observe that when $i = 0$, we are omitting the loop, when $i = 1$, we have the original string in which we traverse the loop a single time, and for all other values of i, we are traversing the loop multiple times.)

However, there are two more conditions that need to be stated in order to make this mathematical description correct. One is that $|uv| \leq n$. When you think about it, this condition is quite obvious. After all, the substring (v) that corresponds to the loop must occur within the first n characters of the string. Otherwise, we obviously haven't decomposed the string s into u, v and w properly. Similarly, $|v| > 0$. This condition is also quite self-explanatory. After all, there must be at least one character in the string which corresponds to the loop, because the shortest possible loop is a self-loop on state Q_k. In case you're having trouble understanding any of this, don't worry. Getting bogged down by mathematical notation is not uncommon. Just pick up your pen and paper, read the above paragraphs again (multiple times, if need be) and draw some diagrams to help yourself think about this and develop some intuition regarding this concept.

Let us now summarise the above discussion with a formal expression of the pumping lemma for regular languages:

The Pumping Lemma (for Regular Languages):

Suppose that L is a language accepted by a finite automaton M. If n is the number of states of M, then for every string $s \in L$ satisfying $|s| \geq n$, there are three strings u, v and w such that $s = uvw$ and the following three conditions are true:

(1) $|uv| \geq n$ (2) $|v| > 0$ (3) $uv^i w \in L$ for all $i \geq 0$

Now that we know what the pumping lemma is and what property it claims that all regular languages have, let us return to our original objective. We wanted a way to show that it is simply not possible to design a finite automaton for accepting certain languages, so that, given a task to design an automaton for a particular language, we either design it (if it is possible), or prove mathematically that is it not possible to do so because the given language is not regular (and that not being able to design an automaton for it isn't a skill issue).

So, how do we actually use the pumping lemma to achieve our goal? We do so using a proof by contradiction. We assume that the given language is regular. If it is regular, then it must satisfy the pumping lemma. However, we show that it doesn't, which produces a contradiction, thereby invalidating our initial assumption that the language is regular. (And since the language is irregular, it is not possible to design a finite automaton for it, thus getting us out of the impossible task of doing so.) Now, the question becomes: how do we show that a certain language does not satisfy the requirements of the pumping lemma? To answer that, let us first discuss an overview of the procedure and then move on to some concrete examples to further clarify the procedure.

First of all, observe that the pumping lemma makes a claim about all strings of the language in question having $|s| \geq n$. Since the claim is universal, we need to show only a single counter-example to disprove it. Thus, we have the liberty of choosing the string on which to work. However, the string must be chosen in terms of n, because otherwise, we cannot be sure that it meets the condition $|s| \geq n$. In order to make the procedure clear, let us continue with the example we gave in the very first paragraph, i.e. $L = \{0^i 1^i \mid i \geq 0\}$. For this language, let us choose the string $s = 0^n 1^n$. Note that we could not have chosen a fixed string like 01, 0011, $0^{100}1^{100}$ or anything like that, because then we could not be sure that the length of the string is greater than or equal to n (the number of states in a hypothetical machine that would accept this language—the very thing we are trying to prove does not exist).

Also, the choice of string you make here matters. For some strings, the property described by the pumping lemma does hold, even if the language is not regular, because all we need for a language to be irregular is for the lemma to not hold for a single string in that language, even if it holds for all others. So, if you choose a string for which you are unable to show that the pumping lemma is violated, try again with a different one. With practice, you will soon be able to develop an intuitive feel about this and make the right choice within the first try (or at least within the first few attempts).

Now that we have chosen a particular string in terms of n (having $|s| \geq n$) from the language L, we need to show that decomposing it into three substrings u, v and w produces a contradiction between the three conditions described in the formal statement of the pumping lemma on the previous page, i.e. (1) $|uv| \geq n$, (2) $|v| > 0$, and (3) $uv^i w \in L$ for all $i \geq 0$. In particular, we show that if the first two conditions hold, the third one doesn't, or, in other words, pumping the string results in strings that are not necessarily in L. (Pumping is the process of omitting or repeating the substring v, i.e. changing the value of i.)

There is something very important to be noted at this point: here, the pumping lemma does not make a universal claim regarding the decomposition of s into u, v and w. It does not state that for all possible decompositions of s into these three substrings, the three conditions described necessarily hold. It claims only that there is some possible decomposition of s which possesses this property. As a result, one counter-example is not enough here. We must show that the contradiction in the conditions occurs for every possible decomposition of s into the three components u, v and w. Now this might seem like a daunting task at first as there are innumerable ways s could be decomposed into three substrings, but don't worry. We merely need to carry out the decomposition in a generic manner, and doing so will automatically handle all possible decompositions.

For instance, for the particular example we are discussing, let us decompose $s = uvw$ such that $u = 0^{p-k}$, $v = 0^k$ and $w = 1^p$, where $0 < k \leq p$. Note how we decomposed the string in way that automatically includes all possible decompositions that fulfil the conditions $|uv| \leq n$ and $|v| > 0$. Take particular note of the range of values of k in this regard. Go on, take a moment. Now that we have carried out the decomposition step in such a way that the first two conditions are fulfilled, all we need to do is show that the third condition does not hold (i.e. pumping the string takes it out of the language L) and we shall have a contradiction on our hands, thereby proving that the pumping lemma does not hold for this language and so it is not regular (and a finite automaton accepting it cannot be made).

With regard to showing the problem with pumping the string, once again, we only need to show a single value of i for which uv^iw is not in L, because the lemma's claim here is that $uv^iw \in L$ for *all* $i \geq 0$. So, a single counter-example will suffice here. For the example under discussion, let's put $i = 0$. Then our string become $s' = uv^0w = uw = 0^{p-k}1^p$. Since the number of 0's and 1's is no longer equal, the pumped s' is not a member of the language L. (Recall the definition of L here—it consists of all strings having a certain number (≥ 0) of 0's followed by the same number of 1's, which has now been violated by s'.) It is obvious that any value of $i \geq 2$ would also serve our purpose here (though it is not necessary that so many values of i will work in all examples; sometimes, only a single one works, and sometimes only a few).

Take a few minutes to go over this example again. More than once, in fact. In particular, note that the lemma makes a universal claim about all strings in L, so we counter it for a single string s. Then it makes an existential claim about a decomposition of s so we counter it universally by working with all decompositions in a generic manner. And lastly, it makes a universal claim in terms of the values of i so we counter it using just one particular value of i which violates the claim.

As an aside, for this particular language, note that we are required to store information about the number of 0's in the string, so that we can compare it with the number of 1's that will come later on in the string. As there is no constraint on how large the number can be, it is quite easy to see that we would require a finite automaton having infinite states (which is impossible) to keep track of all the possibilities, because finite automata do not keep track of any information except their current state. Thus, it is obvious that this language is irregular. As we go along, you should try to develop such intuition for the structures of various languages, which will greatly help you determine whether a language is regular or not. You can then direct your efforts accordingly into either designing the appropriate automaton or proving that the language is not regular using the pumping lemma.

The main objective of this article was to provide you with some basic knowledge regarding what the pumping lemma is and how it is used to prove the irregularity of languages (as well as help you develop some intuition for how all of it works). That task is now done. However, in the addendum that follows, I have included some other examples as well. Naturally, they are only a small subset of the possible questions that could be posed from this topic, intended only to help you get into the flow of working with the lemma. Ultimately, it is up to you to do as much practice as you can to hone your skills in the subject of automata as a whole as well as this topic in particular, as it is a bit technical and thus requires a certain degree of reinforcement in order to be able to master. Not many are mathematically inclined, so don't hesitate to reread the parts that you are still not clear about. And if you have already understood it, then well done!

Before we move on to the examples, a few words from my end. (You may skip this section if you so wish.) Firstly, as of the moment I have written this article, I am a student of BS Computer Science at FAST National University of Computer & Emerging Sciences, Lahore, Pakistan, currently studying in my 6th semester. I studied the Theory of Automata course in my 5th semester and found it to be quite fascinating. The reason behind writing this article was to help students develop some intuition regarding this topic the way I did, as the explanations in the various resources I found on it seemed either too superficial or too formal and mathematical. However, being a student myself, I am extremely liable to have made mistakes in it. If you find any issue in this article, or if you have any suggestion regarding how it could be improved, I would be thankful if you would contact me at contact@nullprime.com about it.

Other than that, I must acknowledge that the formal statement of the lemma given above is a paraphrased form of the one given in *Introduction to Languages and the Theory of Computation* (4th Edition) by *John C. Martin*. Also, all the numbers mentioned in this article are naturally whole numbers (0, 1, 2, …). Further, keep in mind that various authors have various notations for the concepts expressed in the article, so don't get confused if you see a different notation. For instance, s could be taken as x, or uvw as xyz, or n as p, etc. or there could be even more significantly different notations. Lastly, the way I decomposed s in the above example technically incorporated only the cases where $|uv| = n$. To handle all cases of $|uv| \leq n$, we would need to have $u = 0^{p-q-k}$, $v = 0^k$ and $w = 0^q 1^p$, where $0 < k \leq p$ and $0 \leq q < p$, but I omitted q to keep the complexity low as it does not actually affect the outcome of the procedure ($+q$ and $-q$ get cancelled to 0).

Addendum

Now, we shall study a few more examples of the use of pumping lemma in disproving the regularity of various languages. Note that you will have to adjust the level of formalism and detail according to the requirements of your course while writing such proofs yourself, because my objective here is not to show you how to present your proofs, just how to do them. For this reason, I have kept the proof statements brief, but provided notes explaining my choices at various points. Without further ado, let's dive into it. (Example 1 was covered previously, so we start the numbering from Example 2 here.)

Example 2:

$$L = \{a^i b^j \mid i < j\}$$

Assume L is regular.
We choose $s = a^n b^{n-1}$. (Note that $|s| = |a^n b^{n+1}| = n + (n+1) = 2n + 1 \geq n$.)
Next, we take $s = uvw$ such that $u = a^{n-k}$, $v = a^k$ and $w = b^{n+1}$, where $0 < k \leq n$.
(Once again, note that $|uv| = (n-k) + k = n$ and $|v| = k > 0$.)
Now, we pump v using $i = 2$:

$$s' = uv^2w = a^{n-k}\left(a^k\right)^2 b^{n+1} = a^{n-k}a^{2k}b^{n+1} = a^{n-k+2k}b^{n+1} = a^{n+k}b^{n+1}$$

As $k > 0$, $n + k \not< n + 1$. Thus, the pumped string $s' \notin L$, so L is not regular.
(Note that if the definition of L had contained $i \leq j$, we could still have followed the same procedure but we would have had to use $i = 3$ at least to ensure $i \not\leq j$.)

Example 3:

$$L = \{1^{i^2} \mid i \geq 0\}$$

Assume L is regular. We choose $s = 1^{n^2}$. (The number of 1's must be a perfect square.)
Next, we take $s = uvw$ such that $u = 1^{n-k}$, $v = 1^k$ and $w = 1^{n^2-n}$, where $0 < k \leq n$.
(Note that $|uvw| = n - k + k + n^2 - n = n^2 = |s|$, $|uv| = (n-k) + k = n$ and $|v| = k > 0$.)
Now, we pump v using $i = 2$:

$$s' = uv^2w = 1^{n-k}\left(1^k\right)^2 1^{n^2-n} = 1^{n-k}1^{2k}1^{n^2-n} = 1^{n-k+2k+n^2-n} = 1^{n^2+k}$$

The next number after n is $n + 1$, so the next perfect square after n^2 is:

$$(n+1)^2 = n^2 + 2n + 1.$$

But $|s'| = n^2 + k$, and since $0 < k \leq n$, we have $n^2 < n^2 + k < n^2 + 2n + 1 = (n+1)^2$
This means that $|s'|$ is between two consecutive perfect squares and so not a perfect square itself. Thus, the pumped string $s' \notin L$, so L is not regular.

Example 4:

$$L = \{a^i b^j \mid i \neq j\}$$

Assume L is regular. We choose $s = a^{n!}b^{(n+1)!}$. (This is valid because $n > 0$, naturally.)
We take $s = uvw$ such that $u = a^{n-k}$, $v = a^k$ and $w = a^{n!-n}b^{(n+1)!}$, where $0 < k \leq n$.
(Note that the odd decomposition had to be done to ensure that $|uv| = (n-k) + k = n \leq n$.)
We need to find an i such that pumping v i times will result in $|a| = |b|$.

$$n - k + ik + n! - n = (n+1)! \Rightarrow (i-1)k + n! = (n+1)n! \Rightarrow (i-1)k = n \cdot n! \Rightarrow i = 1 + \frac{n \cdot n!}{k}$$

As $k \leq n$, it is a factor of $n!$. (As a matter of fact, this is exactly why we included $n!$ in our chosen string.)
Thus, i is an integer which results in uv^iw having $|a| = |b|$, so the pumped string s'.
Therefore, L is not regular.