09 - Non-Regular Languages and the Pumping Lemma

Languages that can be described formally with an NFA, DFA, or a regular expression are called *regular languages*. Languages that cannot be defined formally using a DFA (or equivalent) are called *non-regular languages*.

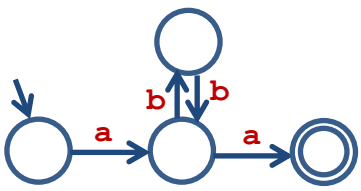
In this section we will learn a technique for determining whether a language is regular or non-regular. To tackle this problem, first note that we only need to concern ourselves with infinite languages – finite languages are always trivial to specify using a regular expression or DFA. This turns out to be useful, because in order for a DFA, which has a finite number of states, to express an infinite language, *it must contain at least one loop*. Let's first take a minute to see why that must be so.

The "Pigeonhole" Principle

A big name for the rather obvious fact that if *n* pigeons are placed into *fewer than n* holes, then some hole must have more than one pigeon in it. This concept turns out to be applicable to regular languages. Here's how:

If a language is regular, then by definition there exists a DFA (or NFA) that describes it. That FA has a finite number of states. Imagine, for example, some language L described by a NFA with 4 states. Now suppose that a particular string (say, "abbbbbba", which contains 8 characters) was known to be a member of language L. Recall that the process of using a FA to recognize a string involves *jumping from state to state as each character is consumed*. It follows that our 4-state NFA *must* contain a loop, otherwise "abbbbbba" would have traversed/exhausted all 4 states in the NFA before reaching the end of the string (contradicting that the string is in L). The furthest it could possibly get would be "abbbbbba" (the portion in red).

But, if our NFA included a *loop*, there would be no limitation on the size of the strings in language L, and our entire "long" string "abbbbbba" could easily be accepted even by a 4-state NFA, such as the following one:



Thus, the existence of a string in L that is longer than the number of states in a finite automata describing L, *necessitates* that if L is indeed regular, then the corresponding automata must include at least one loop (or if you prefer, the corresponding regular expression must contain at least one "`*`").

Now, the existence of a loop has a convenient implication. A loop, by definition, is not limited to being traversed once; it can be traversed an arbitrary number of times. In our above example, we know that abba, abbbba, abbbbbba, abbbbbbbba, and so on, all must *also* be in L. Even executing the loop *zero times* is possible, meaning aa *must also* be in L.

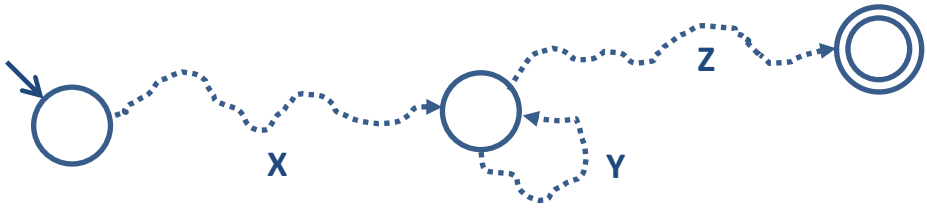
This gives rise to a clever strategy for proving that a language *isn't* regular. Suppose that we have an English description of some language, and we haven't been able to devise a regular expression or NFA for it, and we suspect that it might not be regular. If we can find a ***very long*** string that we know is ***in*** the language, but that string couldn't have been generated by a loop, then the language can't be regular. This strategy is called….

The Pumping Lemma

This theorem describes a property that a language must have in order to be regular. It isn't a "sufficient" condition, meaning that we **can't** use it to prove that a language **is** regular. However, because it is a "necessary" condition, we can use it – in a proof by contradiction - to show when a language is not regular. That is always how the pumping lemma is used.

Briefly, the pumping lemma states the following:

> For every sufficiently long string in a regular language L, a subdivision can be found that divides the string into three segments `x-y-z` such that the middle "`y`" part can be repeated arbitrarily ("pumped") and all of the resulting strings `x-y*-z` are also in L.



"Sufficiently long" means that every regular language has some finite length *p*, called its "pumping length", such that every string with length greater than *p* can be pumped (meaning, has a "loop zone" that can be used to produce other strings that also must be in the language).

The proof of the pumping lemma isn't complicated, but we will concentrate instead on how to *use* it. [incidentally, there is another pumping lemma for context-free languages, but we will skip that].

The Pumping Lemma as an Adversarial Game

Arguably the simplest way to use the pumping lemma (to prove that a given language is non-regular) is in the following game-like framework:

There are two players, Y ("yes") and N ("no"). Y tries to show that L has the pumping property, N tries to show that it doesn't.

1. Y chooses a number *p* and claims it is the pumping length.
2. N chooses a string *s* that is longer than *p*, and claims that *s* cannot be subdivided into $x-y-z$ to satisfy the pumping lemma.
3. Y partitions *s* into $x-y-z$, such that $|y|\geq 1$, and $|xy|\leq p$, and claims that y can be pumped repeatedly, with all resulting strings also being in L.
4. N finds a number *i* such that $x-y^i-z$ isn't a string in L, showing that the pumping lemma isn't satisfied and L is non-regular.

If N succeeds in step 4, the grammar is non-regular. If N fails, then the grammar might be regular. There is an inherent assumption that both players make the best possible moves. So to use this method, it is necessary to consider all possible partitions of *s* in step 3.

Applying the pumping lemma takes some practice. Of course we assume that player Y is "bluffing" in steps 1 and 3. Choosing the variable "p" as the pumping length in step 1 is customary because it then works for any length.

String *s* is chosen to be the counter-example on which L fails to be regular (i.e, it cannot be "pumped"). The proof only requires *one* such contradiction.

Example 1 – Show that the language B = { 0^m1^m } is not regular.

proof:

1. Y chooses pumping length = *p*. For this example, any length works.
2. N chooses string *s* = 0^p1^p. Note that by choosing *s* in this manner, we are guaranteed that xy will contain all 0's, because $|xy|\leq p$.
3. Y is required to choose a partitioning xyz where $y=0^k$
4. N chooses any positive $i\neq1$, such as i=2, so $xy^2z = 0^{p+k}1^p \notin B$

Example 2 – Show that the language B = { w | $w=w^R$ } is not regular (here, *R* means "reversed". That is, B is the language of palindromes)

proof:

1. Y chooses pumping length = *p*. Again, any length works.
2. N chooses string *s* = a^pba^p.
3. Y is required to choose a partition xyz where xy is a^k since $|xy|\leq p$.
4. N chooses any positive $i\neq1$, such as i=2, so $xy^2z = a^{p+k}ba^p \notin B$ because such a string cannot be a palindrome.

Example 3 – show that the language B = { 0^m1^r | $m\neq r$} is not regular.

proof:

1. Y chooses pumping length = *p*.
2. This step is hard. At first it seems like anything N chooses can be pumped. For example, if player N chooses a string like 0^p1^{2p} (such as 000111111 where *p*=3) N would simply choose a partition $x=0$ and $y=00$, and the pumped strings would be 0^11^6, 0^31^6, 0^51^6, 0^71^6, etc., all of which are still in B (because $m \neq r$). The trick is finding a string such that *all* partitions have some pumping point where m=r. N chooses string *s* = $0^p1^{p+p!}$ (for reasons which will be clear later).
3. Y is required to choose a partition xyz where y is 0^k since $|xy|\leq p$.
4. Note the "pumped" version of *s* is xy^iz, and equals: $0^{p+(i-1)k}1^{p+p!}$ since at i=0 it is $0^{p-k}1^{p+p!}$, at i=1 it is $0^p1^{p+p!}$, at i=2 it is $0^{p+k}1^{p+p!}$ etc. Player N needs to show that for every value of *k* that player Y might have chosen, there is a pumping point *i* where the number of 0's equals the number of 1's. That is, where p+(i-1)k = p+p! Indeed, by solving for *i*, player N finds that *i* at $\frac{p!}{k}+1$. Since k≤p, *i* will always be an integer.

Additional Discussion

Don't feel badly if you have difficulty understanding or learning to use the pumping lemma – it is not easy! It takes most students literally *weeks* of practice before it starts to become completely clear. Expect to have to practice it *over and over*, and as parts of it become clear, you'll want to review the *entire* process repeatedly as your understanding solidifies.

The "adversarial game" method described above is one of the simplest and most popular approaches, because it focuses on the loop zone (and its properties) without needing to actually list out the entire partitioning of the string into all of its X, Y, and Z zones. This is sufficient for using the proof, and for most students is the simplest and most direct approach.

However, it isn't the only way of using the pumping lemma. A few students benefit from seeing the *entire* partitioning of the selected string in precise and complete detail. For instance, **example 1** (above) could be expanded in complete detail by partitioning the selected string thoroughly, as follows:

First, we still select the same string $s = 0^p1^p$

Next, we note that partitioning string s into zones X, Y, and Z *must* take the following form, according to the pumping lemma:

0^J	$0^{i \cdot K}$	$0^{p-K-J}1^p$
zone **X**	zone **Y**	zone **Z**

where i=1, K≥1, J≥0. Explained: zone X *may* contain some of the zeros, or may be empty. Zone Y, the loop zone, must contain at least one zero and can contain only zeros; we assume it has looped one time (i=1). Zone Z *may* also contain some of the remaining zeros not in zones X or Y (or it might not). Note that there are exactly P zeros, because J+K+P-K-J = P.

Finally, we "pump" zone Y by setting i=2, producing the following string:

$s' = 0^{J+(2K)+(p-K-J)}1^p = 0^{K+p}1^p$ which is not in language B.