# Lecture 4.5: The Pumping Lemma, Revisited

Ryan Bernstein

April 12, 2016

I got the sense that some people may have been a little bit lost during Tuesday's lecture on non-regular languages. This is more than okay — as mentioned previously, this is my first time teaching the class, so if a lot of people are lost, the problem is likely on my end. The Pumping Lemma is also probably the third most difficult concept we'll be covering in this class, and the second is another version thereof.

With that in mind, I've decided to take a little bit of time to revisit it, and specifically, to address some questions regarding the meaning and value of the pumping length $p$.

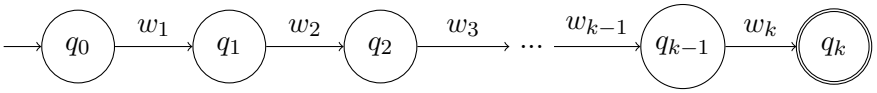
## 1 A Brief Review of the Pumping Lemma

If a language $A$ is regular, then there exists some length $p$ (know as the *pumping length* such that every string $w \in A$ with a length greater than $p$ can be divided into three parts $w = xyz$ that satisfy the following properties:

1. $xy^iz \in A$ for all $i \geq 0$
2. $|y| > 0$
3. $|xy| \leq p$

## 2 Loops and the Pumping Lemma

The Pumping Lemma is based on the observation that a DFA *must* execute some loop or cycle when processing some sufficiently long string. This is because we know that if the string $w$ contains $k$ characters, it will execute exactly $k$ transitions when processed by a deterministic machine. Simply adding states around these transitions as appropriate creates a linear sequence of $k + 1$ states:




Since we know that a string of length $k$ travels through $k+1$ states, we know that the sequence generated for a string of length $|Q|$ will be $|Q| + 1$ states in length.

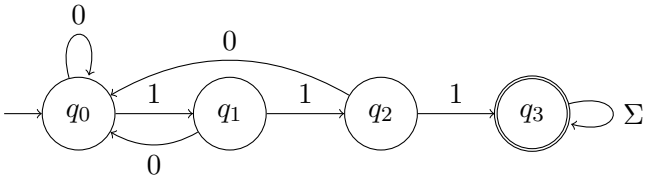
Since this sequence contains more states than are in the DFA in question, the Pigeonhole Principle tells us that at least one state *must* appear in the sequence more than once.

## 2.1 Differentiating $p$ from $|Q|$

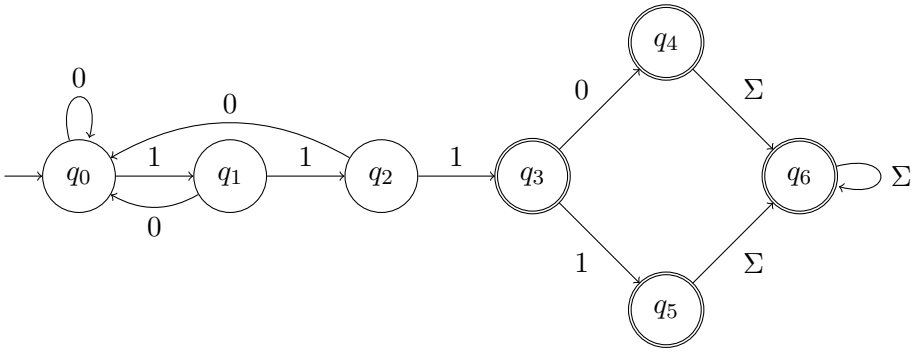
If this is the logic that tells us that sufficiently long strings in some language must traverse a loop or cycle, why introduce $p$ at all? Why not simply use $|Q|$?

The Pumping Lemma holds for all regular *languages*, but $|Q|$ is a property of a *machine* — in other words, $|Q|$ is entirely implementation-dependent.

Consider the language $A = \{s \in \{0,1\}^* \mid s \text{ contains at least 3 1s in a row}\}$. We could draw a minimal DFA for this as follows:

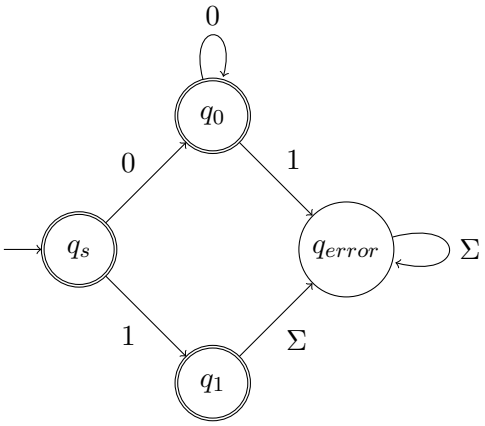



We know that any string of length four or longer will experience some repeated state when processed by this DFA, since the machine contains only four states. However, the following DFA is equally correct:




I can add as many states as I wish, so long as they don't cause a previously accepted string to be rejected or vice versa.

There are also cases like the language $B = 0^* \cup \{1\}$, for which the following DFA can be constructed:




We've again drawn a DFA with four states, but now, any string in $B$ with a length of at least two has a loop in its state sequence.

It's clear, then, that $p$ and $|Q|$ are not always equal. What we *can* say is this: if we constructed a minimal DFA $M$ that decided some regular language $L$ using the smallest possible number of states, then $|Q_M|$ would be an *upper bound* for the pumping length of $L$.

## 3 Applying the Pumping Lemma

We've just spent quite a bit of time deriving the conclusion that we have no idea what $p$ is for a given language. This is especially true since we use the Pumping Lemma to prove that languages are *not* regular, which means that $p$ may not exist at all.

How, then, are we able to use it? The truth is that we don't actually *need* to know the value of $p$. All we need to know (or rather, assume for an indirect proof) is that $p$ exists. When we use the Pumping Lemma to demonstrate that $\{0^n1^n \mid n \geq 0\}$ is a non-regular language, we use the Pumping Lemma with the string $s = 0^p1^p$. I don't need to know the actual value of $p$ to know that $|0^p1^p| > p$ (or to be more precise, that $|0^p1^p| = 2p$).

I also don't know what $x$, $y$, or $z$ look like. All I know is that if $A$ is regular, then $x$, $y$, and $z$ conform to the constraints given in the Pumping Lemma. Since the Pumping Lemma says that *any* string in $A$ longer than $p$ should be divisible into $x$, $y$, and $z$, though, I can choose any string that I feel is convenient. Choosing $0^p1^p$ ensures that however long $y$ is, it contains *only* zeros, since it must appear during the first $p$ characters (all of which are zero).