Pumping Lemma without the Pain

The Pumping Lemma is often half-understood and hence dreaded. The reason it gets hard is that it is not broken into multiple small pieces so that students can check off their understanding. This is a presentation where we will present each nugget in a separate section, and ask you to check-off your understanding. If you want a gentle introduction first, watch an approximately 25-minute Youtube video on it at `https://youtu.be/y9XLtsGCYCk`. (Note: A better video might replace this one.) These notes attempt to make the arguments in that Youtube video rigorous. We begin with a refresher on the basics of logic that we shall need.

1 Propositional and Predicate Logic Refresher

In our presentation, we will be using mathematical logic for precisely stating our results. Here is a refresher, at least for the results we will need:

1. Contrapositive: For Booleans a and b, $(a \Rightarrow b) \equiv (\neg b \Rightarrow \neg a)$
2. DeMorgan's Laws: For Booleans a, b, c, we have:
   (a) $\neg(a \wedge b \wedge c) \equiv (\neg a \vee \neg b \vee \neg c)$
   (b) $\neg(a \vee b \vee c) \equiv (\neg a \wedge \neg b \wedge \neg c)$
3. Forall: $\forall x \in Nat : P(x)$ is equivalent to $P(0) \wedge P(1) \wedge P(2) \wedge \ldots$. Similar results apply for other domains than Nat
4. Exists: $\exists x \in Nat : P(x)$ is equivalent to $P(0) \vee P(1) \vee P(2) \vee \ldots$
5. Negating Forall: $\neg(\forall x : P(x)) \equiv \exists x : \neg P(x)$

   This result can be established through the aforesaid expansions of $\forall$ and $\exists$, and appealing to DeMorgan's Laws
6. Negating Exists can be done similarly: $\neg(\exists x : P(x)) \equiv \forall x : \neg P(x)$
7. Negating Implication: $\neg(P(x) \Rightarrow Q(x)) \equiv (P(x) \wedge \neg Q(x))$

   This result can be established by viewing $a \Rightarrow b$ as $\neg a \vee b$ and applying negation using DeMorgan's laws
8. ("Stack of or to implies" transformation) $(\neg a \vee \neg b \vee \neg c \vee d) \equiv ((a \wedge b \wedge c) \Rightarrow d)$

   This result can be established by expanding the $\Rightarrow$ on the right-hand side and arriving at $\neg(a \wedge b \wedge c) \vee d$, and then applying DeMorgan's law to obtain the left-hand side
9. An implication of the form $(a \wedge b \wedge c) \Rightarrow d$ is made true by making a, b, c true and then making d true. (If we make any of a, b, c false, the implication is vacuously true, and so these combinations need not be considered.)

2 Facts about regular and non-regular languages

All languages are subsets of Σ^*. Regular languages stand out because they have a simpler structure, such as having some number of 0's and some number of 1's. Non-regular languages are more "refined" in the sense that they care even more about what arrangements of 0's and 1's are permitted. A non-regular language may say 0s followed by 1s, but also demand other things—for instance, that their numbers be the same.

Here are additional facts and definitions:

1. A regular language L is one for which there exists a DFA.

2. A *finite* language is L regular. This really says that a language with a finite number of strings is regular.[1]

   This can be shown as follows: One can build one DFA per string in L, and take the union of these DFA.

3. There are also regular languages with an infinite number of strings. In fact, the language Σ^* is regular, and is infinite in size (contains all strings).

4. Even the empty set $\emptyset$ is a regular language. It can be rendered as a DFA that has one initial state q_0 and no other states at all (and no final states at all)

5. A non-regular language *must* be infinite. This follows from the fact that all finite languages are regular.

   This allows us to conclude that A non-regular language has strings of *any* length. In other words, strings in a non-regular language are not bounded in length.

[1] Keep in mind that we only consider finite strings always.

3 Pumping Lemma: Just one of those theorems *about regular languages*

The Pumping lemma happens to be just one theorem pertaining to regular languages. There are many Pumping Lemmas pertaining to regular languages. Most of them are of the form $Reg(L) \Rightarrow Cond(L)$ where Reg (abbreviation for "regular" and $Cond(L)$ is an abbreviation for "condition involving language L") is a predicate.[2]

One of the simplest of regular language Pumping Lemmas (the one we shall study) is:

$$
\begin{array}{l}
Reg(L) \Rightarrow \\
\quad \exists N \in Nat : \\
\quad \forall w \in L \ : |w| \geq N \\
\qquad\qquad \Rightarrow \\
\qquad\qquad\quad \exists x, y, z \in \Sigma^* \ : \\
\qquad\qquad\quad \wedge \qquad w = xyz \\
\qquad\qquad\quad \wedge \qquad |xy| \leq N \\
\qquad\qquad\quad \wedge \qquad y \neq \varepsilon \\
\qquad\qquad\quad \wedge \qquad \forall i \geq 0 \ : \ xy^i z \in L.
\end{array}
$$

It has the form $Reg(L) \Rightarrow Cond(L)$ where $Cond(L)$ is the long formula beginning at $\exists N$. While perhaps intimidating, it expresses something simple:

1. There is $N \in Nat$ such that
2. For all $w \in L$ longer[3] than N
3. w can be viewed as a concatentation of three strings x, y, and z
4. such that the length of xy is $\leq N$
5. and y is non-empty
6. and
   (a) repeating y in xyz ("pumping up"), or
   (b) removing y from xyz ("pumping down")

   results in strings that still are in L

[2]There are also Pumping lemmas of the form $Reg(L) \Leftrightarrow Cond$ which are called "if and only if" Pumping lemmas, or complete Pumping lemmas. We won't be studying them in this course.

[3]What if the DFA has no $w \in L$ that is longer than N? Then the implication in $Cond$ is trivially satisfied. Nobody would shed any tears here. Thus, PL is true even of wimpy languages such as $\emptyset$ that are regular.

4 Why is PL true of regular languages?

You may safely skip this section during your first read.

PL, standing for "The Pumping Lemma" (repeated below) has a simple proof:

$$
\begin{array}{l}
Reg(L) \Rightarrow \\
\quad \exists N \in Nat : \\
\quad \forall w \in L \ : |w| \geq N \\
\qquad\qquad \Rightarrow \\
\qquad\qquad \exists x, y, z \in \Sigma^* \ : \\
\qquad\qquad \wedge \quad w = xyz \\
\qquad\qquad \wedge \quad |xy| \leq N \\
\qquad\qquad \wedge \quad y \neq \varepsilon \\
\qquad\qquad \wedge \quad \forall i \geq 0 \ : \ xy^i z \in L.
\end{array}
$$

- If L is regular, it has a DFA , say of N states
- If you take a string w of length N or above that is in L, then w must describe a path beginning at the initial state q_0 of D and end at some final state $f \in F$ of the DFA D
- But because w is N-long and the DFA D has also N states, then we can't be visiting distinct states along the "journey" xyz. Some two states q must repeat in this journey
- A repeating state means that the journey visits q, then moves away from q, and then re-encounters q, and then moves on (perhaps re-encountering q which we don't care), and lands at f.
- In fact, we must be able to find the "fold" (or loop) described by y within the first N steps of the DFA. Even there, the pigeon-hole principle kicks in, demanding that two states be the same. This explains the "xy within N steps" part of PL ($|xy| \leq N$)
- Because we encounter q and then re-encounter it, y (the looping part) cannot be empty (thus, $y \neq \varepsilon$)
- Now, if the DFA has an encounter with q and then the first re-encounter with q, we can infer that this DFA has many other paths (we will call them "leaky paths")
  - Don't take the path leading to the first re-encounter. Instead, move directly from q toward F, skipping the loop that causes the re-encounter. This is the $i = 0$ case.
  - Take the loop that re-encounters q, but then take this loop once more, twice more, etc (the $i = 2, 3, \ldots$ case)

5 What can we do with the PL?

Since it is a theorem pertaining to regular languages, your first tendency may be to plug in regular languages in place of L and see $Cond(L)$ becoming true. After a few hours of doing so, you'll be bored to tears, because $Cond(L)$ will always come out true. It is a theorem after all.

What else can we do with PL? Aha, hidden within every implication is a contrapositive form! That is, if $Reg(L) \Rightarrow Cond(L)$, then certainly $\neg Cond(L) \Rightarrow \neg Reg(L)$. This gives you some cheer, as you can now now *try and prove* some languages to be non-regular (this being the main use of PL).

If we can plug in place of L a *suspected* (but not actually demonstrated) non-regular language, say L_1 and indeed $\neg Cond(L)$ emerges true, then we can **confirm** (get a proof) that L_1 is non-regular. That is, establishing $\neg Cond(L_1)$ is true allows us to conclude $\neg Reg(L_1)$. We say "try and prove" because with this simplest of PLs, we may not succeed in proving that $\neg Cond(L)$ is true.[4]
The negated condition $\neg Cond(L)$ can be systematically obtained (if necessary, refer to the logic refresher presented earlier):

$Cond(L)$ **is as follows:**

$$
\begin{array}{l}
\exists N \in Nat : \\
\forall w \in L \ : [\ |w| \geq N \\
\qquad\qquad \Rightarrow \\
\qquad\qquad\quad \exists x, y, z \in \Sigma^* \ : \\
\qquad\qquad\quad \wedge \qquad w = xyz \\
\qquad\qquad\quad \wedge \qquad |xy| \leq N \\
\qquad\qquad\quad \wedge \qquad y \neq \varepsilon \\
\qquad\qquad\quad \wedge \qquad \forall i \geq 0 \ : \ xy^i z \in L \ ].
\end{array}
$$

$\neg Cond(L)$ **is as follows**

(keep the $|\ w\ | \geq N$ un-negated; turn implication into conjunction; negate the consequent of the implication):

$$
\begin{array}{l}
\forall N \in Nat : \\
\exists w \in L \ : [\ |w| \geq N \\
\qquad\qquad \wedge \\
\qquad\qquad\quad \forall x, y, z \in \Sigma^* \ : \\
\qquad\qquad\quad \vee \qquad w \neq xyz \\
\qquad\qquad\quad \vee \qquad |xy| > N \\
\qquad\qquad\quad \vee \qquad y = \varepsilon \\
\qquad\qquad\quad \vee \qquad \exists i \geq 0 \ : \ xy^i z \notin L \ ].
\end{array}
$$

[4]Then please ask for a stronger version of PL, or apply a regularity-preserving transformation to the language and then try and prove *that* language to be non-regular.

6 Proving Languages to be Non-Regular

If we can show $\neg Cond(L_1)$ for some language L_1, we would have shown it to be non-regular. The negated $Cond(L)$ described below must be made *true*. We carry over $\neg Cond(L)$ from the last page, and then provide the transformed form of $\neg Cond(L)$ on the right-hand side (using the "Stack of or to implies" transformation in the logic refresher). We will use the right-hand side form in our proofs.

$$
\begin{array}{l}
\forall N \in Nat : \\
\exists w \in L \ : [\ |w| \geq N \\
\qquad \wedge \\
\qquad\quad \forall x, y, z \in \Sigma^* \ : \\
\qquad\quad \vee \quad w \neq xyz \\
\qquad\quad \vee \quad |xy| > N \\
\qquad\quad \vee \quad y = \varepsilon \\
\qquad\quad \vee \quad \exists i \geq 0 \ : \ xy^i z \notin L \ ].
\end{array}
\quad \equiv \quad
\begin{array}{l}
\forall N \in Nat : \\
\exists w \in L \ : [\ |w| \geq N \\
\qquad \wedge \\
\qquad\quad \forall x, y, z \in \Sigma^* \ : \\
\qquad\quad (\ \wedge \quad (w = xyz \\
\qquad\quad \wedge \quad |xy| \leq N \\
\qquad\quad \wedge \quad y \neq \varepsilon) \\
\qquad\quad \Rightarrow \quad \exists i \geq 0 \ : \ xy^i z \notin L \ ].
\end{array}
$$

Let $L_1 = \{0^i 1^i : i \geq 0\}$ be our first test-language, to try and prove non-regular.

1. We have to make the $\forall N$ true.
2. This means we must find a w in L_1, no matter what N is. Recall that non-regular languages are infinite sets, and so we will indeed succeed in finding $w \in L_1$ where w is $\geq N$ in length.
3. We can pick a w, since it is under a $\exists w \in L$ clause
4. We choose $w = 0^N 1^N$, where N is the length of the DFA involved
5. We must now also make the $\forall x, y, z$ part true. Thus **all splits of w are to be picked.**
6. We must
   (a) view w as xyz
   (b) where $|xy| \leq N$ and $y \neq \varepsilon$.
      i. Go ahead and consider **all** xyz split of w
      ii. We make the following items in the antecedent true: $w = xyz$, $\mid xy \mid \leq N$, and $y \neq \varepsilon$:
         A. Under *all* x, y, z splits of w, y still can contain only 0's.
         B. x could be any string of 0's or ϵ
         C. z could have some "left-over" 0's and then the 1's
      iii. Now $i = 0$ or $i > 1$ (all these i choices) gives us an $xy^i z$ outside of L_1
         A. $i = 0$ gives $0^{N-1}1^N \notin L_1$
         B. $i = k$ where $k > 1$ gives $0^{N+k-1}1^N \notin L_1$
         C. We just need one i. Pick any.
         D. Notice that $i = 1$ is within L_1 (these are the "exact paths" as in our Youtube video). The other is give you the "leaky paths" as in our video
   (c) Hence we have made $\neg Cond(L)$ true, and hence shown $\neg Reg(L_1)$

7 Examples from JFLAP's PL Tutor

In all these cases, if a language is regular, build a DFA; else show it is not regular.

In this page, let's try and identify languages to be regular or not, "by inspection." Then, beginning the next page, we will show our actual work.

1. $L_1 = \{0^i 1^i : i \geq 0\}$
2. $L_2 = \{w \in \{a,b\}^* : \#_a(w) < \#_b(w)\}$
3. $L_3 = \{(ab)^n a^k : n > k,\ k \geq 0\}$
4. $L_4 = \{a^n b^k c^{n+k} : n, k \geq 0\}$
5. $L_5 = \{a^n b^l c^k : n > 5,\ l > 3,\ k \leq l\}$
6. $L_6 = \{a^n :\ even(n)\}$
7. $L_7 = \{a^n b^k :\ odd(n) \text{ or } even(k)\}$
8. $L_8 = \{bba(ba)^n a^{n-1} : n \geq 0\}$
9. $L_9 = \{b^5 w : w \in \{a,b\}^*,\ 2\#_a(w) = 3\#_b(w)\}$
10. $L_{10} = \{b^5 w : w \in \{a,b\}^*,\ (2\#_a(w) + 5\#_b(w)) \bmod 3 = 0\}$
11. $L_{11} = \{b^k (ab)^n (ba)^n : k \geq 4,\ n \geq 1\}$
12. $L_{12} = \{(ab)^{2n} : n \geq 1\}$
13. $L_{13} = \{a^i b^j c^k : \text{if } (i = 3) \text{ then } (j = k)\}$

Answers: (We will have you work these out in class and/or do some in your assignments.)

8 A second example

If we can show $\neg Cond(L_3)$ for some language L_3, we would have shown it to be non-regular. This negated $Cond(L)$ shown below must be made *true*:

$$
\begin{array}{l}
\forall N \in Nat : \\
\exists w \in L \ : [\ |w| \geq N \\
\qquad\qquad \wedge \\
\qquad\qquad \forall x, y, z \in \Sigma^* \ : \\
\qquad\qquad (\ \wedge \qquad (w = xyz \\
\qquad\qquad\ \ \wedge \qquad |xy| \leq N \\
\qquad\qquad\ \ \wedge \qquad y \neq \varepsilon) \\
\qquad\qquad\ \ \Rightarrow \qquad \exists i \geq 0 \ : \ xy^i z \notin L \ ].
\end{array}
$$

Let $L_3 = \{(ab)^n a^k : n > k,\ k \geq 0\}$ be our second test-language, to try and prove non-regular.

1. We have to make the $\forall N$ true.
2. This means we must find a w in L, no matter what N is. Choose $w = (ab)^N a^{N-1}$, where N is the length of the DFA involved. This string is in L_3.
3. We must now also make the $\forall x, y, z$ part true.
4. We must
   (a) view w as xyz
   (b) where $|xy| \leq N$ and $y \neq \varepsilon$.
      i. Go ahead and consider **any** xyz split of w
      ii. So long as $|xy| \leq N$ and $y \neq \varepsilon$, we will have all these cases:
         A. x is some initial piece of the repetition $ababab\ldots$
         B. y is a similar repetition, except it may begin with a b, then perhaps include a whole number of (ab)s, and maybe including an ending a.
         C. z may begin with a b, then some number of "left-over" (ab) iterates, and then include a^{N-1}.
      iii. Now $i > 1$ does not work, as y may be a whole number of (ab)s, and pumping up only increases their number, keeping the string within L_3.
      iv. But since we need only $\exists i$ to be made true, set $i = 0$.
      v. Then the y piece gets pulled out.
      vi. In all the possibilities of y listed above, so pulling out y will result in
         A. A whole (ab) dropping out
         B. An a dropping out (in case y is just one a)
         C. A b dropping out (in case y) is just one b)
         D. Or some number of (ab)s and then one a or a b dropping out

E. In all these cases, the leading pattern won't be in L_3, either by having a "whole (ab) deficit," or a mangled beginning pattern of not having perfect (ab) repetitions

vii. In all these cases, the $\exists i = 0$ meets its obligation of emerging true.

5. Thus, $\neg Cond(L)$ is made true

9 Writing the examples so far at a high level

Having done two examples by actually referring to $\neg Cond(L)$ as a predicate logic formula, we will now write the everyday steps you can follow in explaining your proof. We'll do these two examples again. **This style can be used in presenting your proof.**
L_1:

- Pick a suitable string in L_1 such that it can be "pumped out."
- We pick $w = 0^N 1^N$ where N is the states of the DFA (if one exists) for L_1
- $w = xyz$ will loop at y
- Even though we are required to consider **all the cases of** xyz, we can observe that they consist only of $y = 0$'s
- Thus, by pumping y up (or down), we will obtain a string outside of L_1. (This part will require you to draw a little figure showing this is happening.)
- This shows that we can satisfy $\neg Cond(L)$, and hence we have shown $\neg Reg(L_1)$.

L_3:

- Pick a suitable $w \in L_3$. We pick $(ab)^N a^{N-1}$, keeping in mind that we can pump the y-chunk that will fall within $(ab)^N$ down
- The $w = xyz$ part will lie within the $(ab)^N$.
- Here there are multiple cases for y:
  1. Whole $(ab)^k$ chunks for some k
  2. A b alone
  3. A b followed by whole $(ab)^k$ chunks for some k
  4. Any of the above cases followed by an a
  5. Just an a
- In all these cases, by pumping down, we take-off the y chunk. We no longer have a string of the form $(ab)^n a^k$ for $n > k$. (This part will require you to draw a little figure showing this is happening.)
  - Either the number of (ab)s before a^k goes down
  - Or we break the strict alternation of the (ab)s before the a^k
  - In all these cases, we fall outside of L_3, thus showing that $\neg Reg(L_3)$