The pumping lemma: A roadmap

These notes are based on Sipser section 1.4. Study them to understand what the pumping lemma says, and how it’s used in the problems (these are pretty different things!). Make sure to read Sipser as well, especially example 1.39, 1.40, and 1.42. (You can skip the proof of the pumping lemma, but do study the “Proof Idea.”)

1 Why the pumping lemma is true

This is just a summary, to refresh your memory:

If a language is regular, then there is at least one DFA for it (we can write many more by making small changes, adding useless states, etc.) If we pick one DFA, it has some finite number of states (maybe 5, maybe 500). Let’s call “\(p\)” the number of states it has.

Since our DFA can answer “Accept” or “Reject” for any string, it can process strings with more letters than it has states. The only way it can do that, by the “pigeonhole principle,” is by coming back to some state (or states) more than once. This could involve a small loop, as in a reject or accept state where we just loop in place), or a bigger loop that comes back to a state that was visited before (as in the odd/even automata).

But if a loop can be visited once on the way to an accept state, it can be visited any number of times. So if we have one string that’s accepted and that contains a looped part, we can “pump” the looped part and get an infinite number more strings that are also accepted. That’s exactly what the pumping lemma says.

Note that a language is only guaranteed to have the pumping property if it is regular; otherwise there is no DFA and no loops! If a language is regular, we still don’t know what \(p\) is (and so don’t get to choose it), since we don’t know the automaton that recognizes it.

2 What the pumping lemma says

The pumping lemma says that if a language \(A\) is regular, then any string in the language will have a certain property—provided that it is “long enough,” that is, longer than some length \(p\). We don’t actually know what this length is (and so, we don’t get to choose it), but we know there’s some such number, and we write it as “\(p\)” for easy reference.

The pumping lemma then says that:
• Inside any string in $A$ that's longer than $p$: we can find a piece that can be repeated ("pumped") as many times as we want, and the result will always be in $A$.

• Moreover, this piece can be found within the first $p$ letters of our string.

Note that we don't get to choose $p$ or the repeated piece!

2.1 What's with the $x, y,$ and $z$?

Sipser's statement of the pumping lemma says:

...$s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions: ...

What he means is: we can find a substring in $s$ that can be pumped. We'll call this piece $y$. Then anything before $y$ we'll call $x$, and anything after $y$ we'll call $z$.

Then the whole string can be written as $x$-$y$-$z$ (these are strings, not single letters!). By repeating $y$ zero or more times we get:

$$xz, xyz, xyyz, xyyyz, \ldots xyyyyyyyyyyz, \ldots$$

What the pumping lemma says is that each of these must be in $A$. Note that we don't get to choose $p$ or $y$—we don't know what they might be!

2.2 What do Sipser's three conditions mean?

1. "For each $i \geq 0, xy^iz \in A"$
   Translation: Remember that $xy^2z$ is the same as $xyyz$, etc. So this says that sticking in multiple copies of $y$ will give you strings that are still in the language. For $i = 0$, you get no copies of $y$, i.e., the string $xz$.

2. "$|y| > 0$"
   This is easy: the length of $y$ is not zero, i.e., $y$ is not the empty string. (Otherwise you could claim to be "pumping" any string, by adding a zillion copies of $\varepsilon$!)

3. "$|xy| \leq p$"
   Since $x$ is the piece before $y$, this says that all of $y$ must come from the first $p$ letters of our string $s$ (so that the combined length of $x$ and $y$ can be less than $p$).

**Example:** For a language that *can* be pumped, take the set of strings that begin with 1 and end with 0, with anything in-between.

We don’t have to choose $p$, but you can see that with any string long enough (longer than 3, actually!), you could take any part in the middle of the string and repeat it, and still get something in the language:

For example, we could break $10100010$ like this: $x=1$, $y=01$, $z=00010$. By pumping we then get $xy^0z = 1-00010$, $xy^2z = 1-01-01-00010$, $xy^3 = 1-01-01-01-00010$, etc, all of
which begin with 1 and and with 0. So, the pumping lemma works for this language and this string. (Since the language is regular, it actually works for \textit{all} strings!)

3 How to use the pumping lemma

Here is where things get confusing: the pumping lemma is most useful when we want to show that a language is \textit{not} regular. We use a proof by contradiction: \textit{If} a language is regular, the pumping lemma applies to it, and any string that is “long enough” can be “pumped.” So to get a contradiction:

We need to find one string longer than \( p \), for which there is \textbf{no way} to choose a \( y \) that could be “pumped” (repeated over and over) and still get strings in the language.

The tricky part is demonstrating that there is \textit{no way} for something to happen. If we can do that, it means that the pumping lemma does not apply to our language, so it is non-regular.

So, we can’t choose \( p \) or \( y \), but we \textbf{can} choose the string! Since we don’t know \( p \) we can’t actually write down any specific string and know that it is longer than \( p \), so we give our string schematically, as a formula that contains \( p \) in it.

\textbf{Example:} to show that \( \{ a^nb^n \} \) is not regular, we chose a string consisting of \( p \)-many a’s followed by \( p \)-many b’s, written: \( a^pb^p \).

Remember, \textbf{we} don’t get to choose \( y \)! But since \( y \) has to come from the first \( p \) letters of our string, in this case \( y \) will have to be entirely a’s. And when we pump that, we no longer have the same number of a’s and b’s. So, the resulting strings are \textit{not} in the language we started with! We’ve shown that the pumping lemma does not apply, so this language cannot be regular.

\textbf{Always choose your string to make the first \( p \) letters as similar as possible.}

3.1 What should you actually write?

Your answer must:

(a) explain what you are doing and why.
(b) choose a string that \textit{is in the language} and is long enough.
(c) show that if the string is pumped, some (or all) of the resulting strings are \textit{not} in the language.

Part (a) is always the same. Your job is to choose a string \textit{wisely}, and then to explain part (c).

Remember, you don’t get to choose \( p \) or \( y \)! 
Example: Show that the language \( B = \{ w \mid \text{w has an equal number of 1s and 0s} \} \) is not regular.

Your answer:

I will show that \( B \) is non-regular by contradiction:  

* If \( B \) was regular, the pumping lemma would apply to it.  
* I will choose a string and show that it cannot be pumped.  
* Therefore, the pumping lemma does not apply to \( B \). So \( B \) is not regular.

I choose the string \( s = a^p b^p \), where \( p \) is the pumping length of \( B \). My string is in \( B \), and is longer than \( p \). So the pumping lemma should apply.

Since \( y \) must be chosen from the first \( p \) letters, it will have to be all a’s. Then the string \( xy^2z \) will contain an extra copy of \( y \), so it will be of the form \( a^{p+|y|} b^p \). Since \( |y| \) (the length of \( y \)) is not zero, \( p + |y| \neq p \). So \( xy^2z \) does not contain the same number of a’s and b’s, and so it is not in \( B \). I have shown that the pumping lemma does not apply to \( B \), so it is not regular.