A detailed look at the pumping lemma

Note: *Much of this post explains the idea behind the pumping lemma and tries to warn against some common misconceptions. But in an examination, in a typical proof that uses pumping lemma to show that a language is not regular, you are expected to be formal, brief, and to the point. To see an example of how to write such a proof in the examination, check the last section of this post.*

If you suspect that it is impossible to build a finite state automaton to recognize a certain language, the pumping lemma *may* help you to prove it rigorously. However, it is important to understand the statement of the pumping lemma very precisely to use it correctly.

An Example

Consider the language, L, consisting of precisely those strings over the alphabet $\{0, 1\}$ such that the number of 0s exceed the number of 1s. You suspect that any machine that attempts to recognize the language will have to "remember" the number of 0s and 1s as it is reading the input, and therefore cannot handle it with finite states. However, such an explanation is certainly not enough (definitely not in the examination!!!) and we need a rigorous proof.

The main observation

An automaton works by reading the string one character at a time and depending on the character and its state at the time of reading the character, shifts to a new state. However, an automaton can have only finitely many states, say N states. By the pigeon-hole principle, by the time it reads N characters, it has been in at least one of these states more than once. For example, if the input was 1^N0^{N+1}, even before it starts reading the 0s, it has already repeated a state because the 1s are more than N (the number of states).

Let us look at this more closely: The machine began at the initial state, q_0, but after reading, say l, 1s it ended up in some state q_r which it came back to after reading some more 1s, say m of them. The rest of the string, containing the remaining 1s ($N - l - m$ of them) and the $N + 1$ 0s that follow, takes this q_r to a final state. Since 1^N0^{N+1} belongs to our language, having more 0s than

1s, the final state must be an accept state for the automaton to recognize this string. To summarize:

1. The machine first read 1^l and that took it from state q_0 to q_r.
2. The machine then read 1^m which took it **from state q_r back again to q_r**.
3. The machine finally read $1^{N-l-m}0^{N+1}$ which took it from state q_r to an accept state.

We do not know what l and m are besides the fact that $l + m \leq N$, but that is all the information we will need.

The main observation is this: after it read the 1^l chunk, if I repeatedly fed the machine the chunk 1^m over and over again, it would keep coming back to the same state q_r. The last chunk, $1^{N-l-m}0^{N+1}$, always takes q_r to an accept state, so such a machine is forced to also accept strings of the form $1^l(1^m)^i1^{N-l-m}0^{N+1}$ for *any* i. However, since the part being repeated here consists of only 1s and N is fixed throughout, eventually for some large i, the number of 1s will exceed the number of 0s and result in strings that should not be in our language.

To summarize: if we assumed that an automaton exists to recognize the language, then it must recognize the string 1^N0^{N+1} too where, N is the number of states. The above reasoning then shows us that it must accept certain modifications of this string (repeating some chunk of 1s), many of which do not belong to the language (because the number of 1s are more than the number of 0s). So no automaton can be built to accept precisely those strings that are in our language and reject the rest.

Note how important it was that we chose a string in our language for which the 1s came first and the number of 1s itself exceeded the number of states. That is what guaranteed that the repeated part had only 1s. Had we merely ensured that the entire string was bigger than N, the pumping lemma can promise nothing about the substring responsible for repeating the state. We have to consider the possibility that that substring has only 0s, in which case repeating that chunk will result in strings which are in the language anyway so even though the automaton is forced to accept them, there is no contradiction.

This reasoning generalizes and the generalization is abstracted out in the pumping lemma.

The statement of the pumping lemma

Lemma: If a language can be recognized by a finite state automaton, then there is some number N (called the pumping length) so that if there is a string w in the language whose length exceeds N, then it is possible to view w as a concatenation of three substrings, x, y, and z, i.e. $w = xyz$, so that

1. The middle part, y is not empty (otherwise the lemma would be useless!)

2. x and y are within the first N characters of the string (i.e. $|xy| \leq N$) (this is the only guarantee the lemma provides about where the break up occurs)
3. the language must also contain the modified strings xy^iz for all $i = 0 \ldots$ (because the automaton is forced to accept them too).

Be careful!

You apply the lemma by making a clever choice of w's. However, since you have to account for every possible automaton, you have to be careful that you

1. *Make no assumptions on the pumping length N.* You have to account for the possibility that N could be any natural number and therefore you must consider not one w but a whole sequence of them so that no matter how many states one provides an attempted automaton, there is at least one w with length bigger than that number. That is why we considered *all* strings of the form 1^N0^{N+1} in our example and not just, say $N = 100$. After all, one is claiming that any *hypothetical* implementation cannot really exist, and so we have to assert that no matter how many states one tries to use, it will fail, i.e. it will fail for *all* possible N.
2. *Make no assumptions on where the break up of w into x, y, and z will happen*, except that xy is within the first N characters. You have to account for all the possible ways in which the break up can happen in the first N characters. After all, where the breakup has happened will depend on how one tried to implement the automaton. But we are claiming that no matter how it has been implemented, it will fail. Nevertheless, the fact that it happens within the first N characters can be very useful. In our example, we ensured that not only was the string bigger than N, but the first part containing only 1s was itself bigger than N. This helped us to guarantee that y contained only 1s so that when we repeated y we were increasing the number of 1s while keeping the number of 0s fixed. We could make no assumptions on the number of 1s in y, but luckily we did not need to. All that mattered was that y contained only 1s.

Applying the pumping lemma

If you suspect that language cannot be recognized by a finite state automation, consider a certain collection of strings from your language that have a wisely chosen form so that:

1. For each N there is a string from this collection that has length more than N.
2. You should be able to demonstrate that for *each* of the possible ways of breaking up the string as xyz so that xy is within the first N characters, xy^iz does not belong to your language for at least some i. So your argument

must be general and cannot make any assumptions on x, y, and z other than the fact that $|xy| \leq N$ if the string is viewed in the form xyz.

You would then have shown that no matter how many states an automaton uses, you can always find a string from your language which the automaton must accept, but it is forced to also accept certain modifications of it, some of which do not belong to the language. Therefore, it is impossible to find an automaton that accepts precisely what is in the language but also rejects everything else.

How you should *write* the proof in the exam

Until now we have discussed the idea and thought process behind the pumping lemma. But do not write an essay in the exam! This is how it should be written formally (the footnotes highlight crucial arguments that you must write, otherwise your proof will have gaps and you will lose points):

Exercise Prove that the language consisting of precisely those strings of 0s and 1s, where the number of 0s exceed the number of 1s, cannot be regular.

Proof: If we assume that the language is regular, then let N denote its pumping length. Consider the strings [1] $1^N 0^{N+1}$. For each possible N, the string $1^N 0^{N+1}$ belongs to our language [2] because it has more 0s than 1s. The pumping lemma guarantees that it can be written in the form xyz so that y is non-empty and lies within the first N characters and xy^iz must also belong to the language for each and every natural number i. Since y lies within the first N characters, it involves only 1s [3] and all the 0s lie within z. Therefore, xy^iz will have much more 1s than 0s for large enough i. The pumping lemma says that even such strings must belong to our language, which is a contradiction because the language does not contain *any* string with more 1s than 0s.. Therefore, it is impossible to design a finite state automaton that will recognize our language.

[1] Notice how we are considering not just one string, but a string for each possible N since we do not know what that N may be.

[2] You must explain why each of these belong to the language, otherwise your proof has a gap.

[3] Again, in this case, the crucial observation is that even though we do not precisely know where the y chunk occurs, we know that it has only 1s. Again, this *must* be explained.