The Pumping Lemma Game
CSC 341, “Automata, Formal Languages, and Computational Complexity”
Department of Computer Science · Grinnell College
August 31, 2020

One way to get a handle on logical structure of the pumping lemma for regular languages is to envision it as a two-person game, a ritualized debate between two parties: Pro, an advocate for the thesis that a particular language \( A \) is regular, and Con, who wishes to prove that \( A \) is not regular. Under the rules of this game, Pro gets to choose values for the variables introduced by existential quantifiers and Con chooses values for the variables introduced by universal quantifiers. The order in which they take their turns and make their choices is the same as the order of the quantifiers in statement of Theorem 1.70, except for the first quantifier — we’ll assume that Pro and Con have agreed in advance on the language that is the subject of their dispute.

Here’s the general form of the game play, then:

- Pro chooses the pumping length \( p \).
- Con chooses a member \( s \) of \( A \) such that \( |s| \geq p \).
- Pro divides \( s \) into substrings \( x, y, \) and \( z \) such that \( s = xyz \), \( |y| > 0 \), and \( |xy| \leq p \).
- Con chooses a natural number \( i \).
- A referee checks whether \( xy^i z \in A \). If so, Pro wins; if not, Con wins.

The pumping lemma says that, if \( A \) is in fact regular, then Pro can win this game no matter what choices Con makes. To prove conclusively that \( A \) is not regular, then, Con only needs to win once (assuming that Pro plays as well as possible).

If you read the proof of Theorem 1.70 with this game in mind, you’ll see that it even spells out Pro’s winning strategy when \( A \) is known to be regular: Pro constructs a deterministic finite automaton that recognizes \( A \) and chooses the pumping length to be equal to the number of states of that automaton. Con then comes up with a string \( s \) of length greater than or equal to \( p \). Pro takes this string and runs it through the deterministic finite automaton, keeping track of the sequence of states in the computation. Since the length of this sequence of states is greater than the number of states, at least one state will be repeated. Pro partitions \( s \) by making \( x \) the part of \( s \) that drives the automation to the first occurrence of a state that is eventually repeated, \( y \) the part of \( s \) that drives the automaton from the first occurrence of that state to its second occurrence, and \( z \) the part of \( s \) that drives the automaton from the second occurrence of the repeated state to the accept state at the end of the computation.

This strategy automatically ensures that \( s = xyz \). It also ensures that \( |y| > 0 \), since the automaton must read at least one symbol during the transitions from the first occurrence of the repeated state to the second. Also, \( |xy| \leq p \), since the first repetition must occur within the first \( p+1 \) states in the computation, that is, after no more than \( p \) transitions, consuming no more than \( p \) input symbols. Finally, it ensures that \( xy^i z \in A \) for every natural number \( i \), because \( x \) will drive the automaton from the start state to the first repeated state, and every repetition of \( y \) will drive the automaton from that state back to itself, after which \( z \) will drive the automaton from that repeated state to the final state.

Con has no opportunity to counter this strategy and so always loses when \( A \) is regular.
Examples 1.73 through 1.77 are all about finding strategies for Con to win the game described above by making clever choices of $s$ and $i$. Because Con's chooses $s$ only after Pro has chosen $p$, Con can adaptively respond to Pro's choice. Similarly, Con chooses $i$ only after Pro has divided $s$ into $x$, $y$, and $z$, and in some cases this provides Con with a crucial advantage.