Pumping Lemma (for context-free languages)

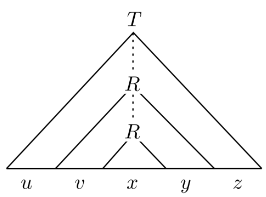
The **Pumping Lemma** for context-free languages gives us a way to prove a language is not context-free. It goes like this:

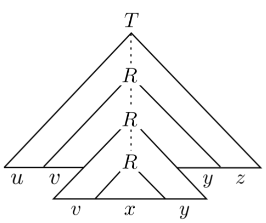
Pumping Lemma: If L is a context-free language, then there is a number p (the pumping length) where, if s is any string in L of length at least p, then s may be divided into five pieces, $s = uvxyz$, such that:

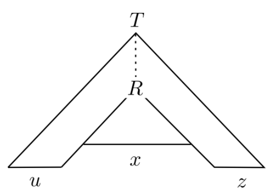
1. for each $i \geq 0$, $uv^ixy^iz \in L$,
2. $|vy| > 0$ (i.e., **either** $v \neq \epsilon$ **or** $y \neq \epsilon$), and
3. $|vxy| \leq p$.

These conditions state that while u or x or z my be ϵ, either v or y (or both) cannot be. Furthermore, it states that s can be pumped (or increased) for each repetition of v or y, but it can also be un-pumped (decreased) when $i = 0$.

Intuition: the key idea is, just like for regular languages (see @49), that the pigeonhole principle guarantees that some <u>sub-tree</u> in a given parse tree must repeat for very long strings and therefore can be pumped. The way to think about this is some rule R must be re-used along a derivation path, thus creating a "loop" (like we saw with DFAs):







<u>Example</u>: L = $\{a^n b^n c^n \mid n \geq 0\}$ is not context-free. <u>Proof</u>: The string $s = a^p b^p c^p$ can be pumped in a way that creates a contradiction. There are a few cases to consider for what v or y can be: one type of symbol, or two types of symbols. When they are one type of symbol, we pump strings that are unbalanced. When they are two types of symbols, we pump strings that could be balanced, but they would be in the wrong order! So, in any case, we pump strings that should, but cannot, be in L, a contradiction.

<u>Example</u>: L = $\{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$ is not context-free. <u>Proof</u>: We work with the string $s = a^p b^p c^p$ again, this time, there are multiple cases and for some we pump-up and others we pump-down. (Rest of proof is left as exercise.)

<u>Example</u>: L = $\{ww \mid w \in$ {0,1}* }. <u>Proof</u>: In creating a contradiction, the obvious try, $s = 0^p 1 0^p 1$, will not work as a contradiction because we could pump things that remain valid:

```
000..000 ... 0  1  0 ... 000..000 1
    u        v  x  y         z
```

But $s = 0^p 1^p 0^p 1^p$ works! There are several cases to consider for where vxy appears: it can be on the left-hand side of ww, the right side, or it can straddle the mid-point. (Proof of left and right cases left as exercises.) If vxy straddles the middle, we pump-down and notice that $0^p 1^i 0^j 1^p$ happens and i and j cannot both be p (by the third condition $|vxy| \leq p$).