Practical Languages

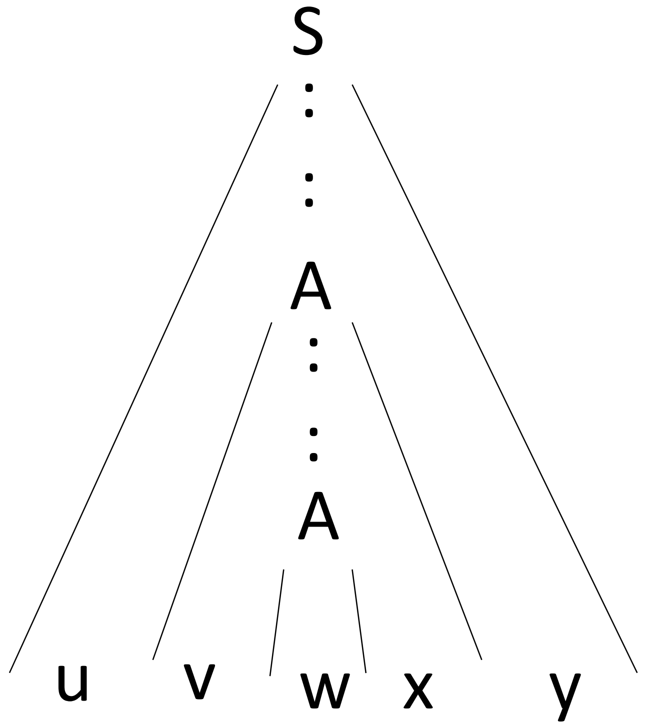
In this section we will consider how the Pumping Lemma applies to standard programming languages.

First, here is an alternative version of the Pumping Lemma. This gives some control over where the pumping strings v and x occur, at the cost of giving up control over the length of string vwx:

An Alternative Pumping Lemma: Let $\mathcal{L}$ be a context-free language. Then there is a constant p so that if z is a string in $\mathcal{L}$ and $|z|>=p$ there must be a decomposition z=uvwxy such that

1) $|u| < p$
2) $vx \neq \varepsilon$ (i.e., both v and x aren't ε)
3) For each i>=0 uv^iwx^iy is in $\mathcal{L}$

To prove this remember our construction for the original Pumping Lemma:



If the string is long enough we can find a variable A that is repeated on some leaf-to-root path; the upper instance of this generates the vwx substring. In the Pumping Lemma we chose A as far down as possible to force |vwx|<=p. Instead, we now choose A to be as far *left* as possible. If u has more than p= 2^N leaves one of them must be of depth greater than N, forcing a repeated variable. If we choose A to be as far left as possible then |u| < p.

Now consider a C-like programming language with the common rule that variables need to be declared before they are used. If this language is context-free let p be its pumping constant. Consider the program at right, with more than p x_i declarations. How might this program be decomposed into uvwxy parts, where v and x can be pumped?

```
int foo() {
        int x0;
        int x1;
        int x2;
        .....
        int xp;
        int s;
        s = (x0+x1+...+xp);
        return ((s+x0+x1+...+xp));
}
void main() {
        print( foo() );
}
```

Note that removing part of any line other than the assignment and return statements in function foo() results in something that is not a program. That fact, combined with our alternative lemma says that the v-portion of a decomposition must remove one or more x_i declarations. The x-portion must remove the corresponding variables from the assignment and return statements.

```
int foo() {
        int x_0;
        int x_1;
        int x_2;
        .....
        int x_p;
        int s;
        s = (x_0+x_1+...+x_p);
        return ((s+x_0+x_1+...+x_p));
}
void main() {
        print( foo() );
}
```

Suppose x_k is one of the variables declared in the v-portion. At the very least the x-portion must include

$s = (x_0+x_1+\ldots+x_k+\ldots+x_p);$

$\text{return } ((s+x_0+x_1+\ldots+x_k+\ldots+x_p));$

Thanks to the parentheses if this portion is removed the result is not a valid statement.

```
int foo() {
        int x0;
        int x1;
        int x2;
        .....
        int xp;
        int s;
        s = (x0+x1+...+xp);
        return ((s+x0+x1+...+xp));
}
void main() {
        print( foo() );
}
```

Alternatively, if the v-portion includes everything from the declaration of x_k through the assignment statement, then because of the placement of s in the return statement the x-portion would need to include the entire return statement.

Any way we slice it, the v and x-portions together need to include the entire assignment statement and the entire return statement. If we pump 0 times these statements are removed and the result is an invalid program (function foo() needs a return statement).

```
int foo() {
        int x0;
        int x1;
        int x2;
        …..
        int xp;
        int s;
        s = (x0+x1+...+xp);
        return ((s+x0+x1+...+xp));
}
void main() {
        print( foo() );
}
```

This shows that our programming language is not context-free. What makes it non-context-free is the need to declare variables before using them. Any construction that makes a statement acceptable if something else was said any arbitrary number of statements previously leads to the language not being context-free. For example, Python doesn't require variable declarations, but the statement x=y+1 is valid if y has previously been assigned to and invalid if it hasn't. A slight variant of our construction can be used to show that the Python language is not context-free.

Our Alternative Pumping Lemma is a special case of *Ogden's Lemma*: Let $\mathcal{L}$ be a context-free language. Then there is a constant p so that if z is a string in $\mathcal{L}$ and |z|>=p, and if we "mark" at least p letters of z, then there must be a decomposition z=uvwxy such that

1) vwx has at most p marked letters
2) vx has at least one marked letter
3) For each i>=0 uv^iwx^iy is in $\mathcal{L}$

We won't prove Ogden's Lemma, though the proof is just a variation on our proof of the Pumping Lemma. Note that we could derive our Alternative Pumping lemma from Ogden by marking the first p letters of z.

We can use Ogden's Lemma to find a language that is pumpable but not context free. Let $\mathcal{L} = \{0^i1^j2^k \mid i, j, \text{ and } k \text{ are all different}\}$. To see that $\mathcal{L}$ is pumpable, consider z in $\mathcal{L}$ where $|z| = i+j+k > 6$. If i is the largest of i,j,k then let d be a number less than i where i-d is neither j nor k. Then let $u = \varepsilon$, $v = 0^d$ $w = 0^{i-d}1^j2^k$, $x = y = \varepsilon$. Then in uv^0wx^0y there are different numbers of 0s than 1s or 2s, and for any n > 0 uv^nwx^ny has more 0 than 1s or 2s. So z is pumpable if 0 is the most common digit. If j is the largest of i,j, and k choose d so that j-d is neither i nor k and let $u=0^i$, $v=1^d$, $w=1^{j-d}2^k$,$x=y=\varepsilon$. Again uv^nwx^ny has different numbers of the three digits for every n. We can do a similar construction of k is the largest of the three digits. So $\mathcal{L}$ is pumpable.

We will use Ogden's Lemma to show $\mathcal{L}$ is not context-free. Suppose it is context free; let p be its pumping constant. Consider $z = 0^p1^{p+p!}2^{p+2p!}$ We will mark the p 0s. Now consider any decomposition z=uvwxy that satisfies Ogden's Lemma. It is obvious that v and x each contain only 1 digit. One of them contains 0s. The other contains at most 1 of the digits {1,2}. So let d be the number of 0s in vx. For any n uv^nwx^ny has p+(n-1)d 0s. If vx contains no 1s then when n=1+(p!)/d uv^nwx^ny has the same number of 0s and 1s and so is not on $\mathcal{L}$. If vx contains no 2s then when n=1+(2p!)/d uv^nwx^ny has the same number of 0s and 2s and so is not in $\mathcal{L}$. Either way, this string z is not pumpable, so $\mathcal{L}$ is not context free.