Logic and Computability
Problem Set 7: Context-Free Languages!

Please read Handout 3 on using the Pumping Lemma for context-free languages.

Note: Online Prolog interpreters such as SWISH (https://swish.swi-prolog.org/) are available. Some students prefer those environments to running `swipl` locally.

Challenge 1: PDAs in Prolog! [30 Points]

In class, we saw most of the code for the `accepts` predicate for PDAs. The beauty of doing this in Prolog is that `accepts` serves *both* as a definition of acceptance in the language of formal logic (i.e., Prolog) *and* a PDA simulator that we can actually use to run PDAs! *Please do not change* the provided `accepts` predicate.

The code that we saw in class was only missing the `newTape` and `newStack` predicates. Your job is to add those predicates in the `accepts.pl` starter file provided with the instructions. Recall that `newTape(Tape, Symbol, NewTape)` is true iff `Tape` is the list of symbols on the tape beforehand, `Symbol` is the symbol that was just consumed or ϵ, and `NewTape` is the list of symbols on the tape afterwards. For example:

```
?- newTape([s, p, a, m], s, X).
X = [p, a, m].

?- newTape([s, p, a, m], epsilon, X).
X = [s, p, a, m] .
```

Similarly, `newStack(Stack, Pop, Push, NewStack)` is true if and only if `Stack` is the current stack, `Pop` is the symbol at the top of that stack (or ϵ, which is always at the top of the stack!), `Push` is the symbol that replaces `Pop`, and `NewStack` is the resulting stack at the end of all of that. For example:

```
?- newStack([a, b, c, $], a, z, X).
X = [z, b, c, $].

?- newStack([a, b, c, $], a, epsilon, X).
X = [b, c, $] .

?- newStack([a, b, c, $], x, epsilon, X).
false. <-- Oops!  The stack did not have an x at the top so this fails!
```

Your task is to add the `newTape` and `newStack` predicates to the `accepts.pl` file. This will be short, involve NO recursion, and have virtually no weird Prolog ordering issues! Our sample solution is two lines for `newTape` (two rules, each one line long) and four lines for `newStack` (four rules, each one line long). Finally, you can test your code on the provided `pda1.pl` file which describes the first PDA that we saw in Lecture 15 for the language $L = \{0^i 1^i \mid i \geq 0\}$. Here's what you should see:

```
?- [accepts].  <-- load in accepts.pl
true.
?- [pda1].  <-- load in pda1.pl
true.

?- accepts(q0, [0, 0, 0, 1, 1, 1], []).
true

?- accepts(q0, [1, 0, 0, 1, 1, 1], []).
false.
```

For even more testing fun, test your code on the provided `pda2.pl` which describes the PDA that we saw in Lecture 15 for the language $L = \{ww^R \mid w \in \{0,1\}^*\}$. Submit the file `accepts.pl`.

Challenge 2: Constructing PDAs [30 Points]

Now, construct your own PDA for the language $L = \{0^{2i} 1^i \mid i \geq 0\}$ in Prolog in a file called `problem2.pl`. (Ideally, you should test your code with the `accepts` predicate from Problem 1, but you can also encode the PDA in this file without using the results from Problem 1 to test it.) Please name your start state `q0`. Here's what it should look like:

```
?- [accepts].
true
?- [problem2].
true

?- accepts(q0, [0, 0, 0, 0, 1, 1], []).
true

?- accepts(q0, [0, 0, 0, 0, 0, 1, 1, 1], []).
false
```

Note that the syntax of a PDA allows us to only pop a single symbol (or ϵ) and only push a single symbol (or ϵ). If you wish to push or pop more than one symbol in a given transition, you'll need to find a way to do that within rules of the PDA syntax. Don't modify your `accepts` to try to handle pushing or popping multiple symbols in one step. We will be testing your code using our own `accepts` predicate which adheres to the official definition of a PDA.

Challenge 3: Proving that Languages are not Context-Free [40 Points]

Use the Pumping Lemma for CFL's to show that the language $\{0^i 1^i 2^j | i \geq j\}$ is not context-free.

Answer 3:

Let us assume for the sake of contradiction that the language $\{0^i 1^i 2^j | i \geq j\}$ is context free. Thus, let n be the value from the Pumping Lemma. Now, let $w = 0^n 1^n 2^m$ such that $n \geq m$. We can see then that $w \in L$ and let u, v, x, y, z be strings such that $w = uvxyz$, $|vxy| \leq n$, and $|vy| \geq 1$. First, we can see clearly that vxy cannot contain 0s, 1s, and 2s, as $|vxy| \leq n$ and our 0s and 2s are seperated by n 1s. Now, we can also see that vxy cannot be only 0s. If it were the case, then we could have that $vxy = 0^k$ for $1 \leq k \leq n$, then as $|vy| \geq 1$ and $|vxy| \leq n$, then it follows that uv^0xy^0z has fewer 0s than 1s, and thus is not in L. For the same reasoning, we can see that if $vxy = 1^k$ for $1 \leq k \leq n$, then uv^0xy^0z has fewer 1s than 0s, and thus is not in L. This is the same case for if $vxy = 2^k$, as then, as n is finite, there exists some integer, call it m such that when we compute uv^mxy^mz, we obtain more 2s than 1s or 0s, as we can see that we can just add 2s until we break this condition. Thus uv^mxy^mz is not in L. Now, consider if vxy contains both 0s and 1s and is of the form $0^l 1^p$ such that $1 \leq l \leq n-1$, $1 \leq p \leq n-1$ and $l + p \leq n$. Consider then that it follows that uv^2xy^2z must have either more 0s than 1s or more 1s than 0s. We can see that this is true as if choose $p = l$, then it follows that uv^2xy^2 will subsequently have an uneven number of 0s and 1s, as x must be a 0 or a 1, and thus when we raise v, y to the power of 2, we end up more 1s than 0s or more 0s than 1s, since v would be of the form 0^{p-1} when $x = 0$ or 0^p, when $x = 1$, either way resulting in uv^2xy^2z being a string is not in L. Now, considering if $p \neq l$, then it follows that the string uv^2xy^2z will have either more 1s than 0s or more 0s than 1s due to the fact that however we break up the string, it follows that when we exponentiate, then the difference between p and l increases, and thus, whichever is larger, p or l, will produce more 0s or 1s, thus leading to uv^2xy^2z not being in our language L. Thus, it follows that $uv^2xy^2z \notin L$. Now, consider if $vxy = 1^l 2^p$ such that $1 \leq l \leq n-1$, $1 \leq p \leq n-1$ $l + p \leq n$. We can see evidently that if we have uv^0xy^0z, then it follows that we are going to have less 1s, since $|vy| \geq 1$, than 0s, since we are removing some number of 1s, but retaining our original number of 0s. Thus, we can see that $uv^0xy^0z \notin L$. In each case we have a contradiction, and thus L cannot be context-free.