Assignment 5

Alex Clemmer
Student number: u0458675

Problem 1:

`0, 011, 111`

Problem 2:

This PDA accepts all strings except those with only one `1`.

Problem 3:

a)

The basis of the automaton is that it will "guess" when the middle `a` of the string is. But there are a few other challenges as well.

One thing to keep in mind is that for a string to have a "middle letter", it must be an odd number of letters long. This ends up making the PDA easier to construct, as we will see.

Given that we don't really care about the characters that come before or after the middle `a`, all we need to do is count them and make sure they balance up. So for the first part of the PDA, we will push a symbol to the stack for every character we see, and after the middle `a` we will pop once for every symbol we find. If the stack isn't empty, then we reject.

Those parts are easy; the key is finding the middle of the string, and making sure there's an `a` there. The way to do this is to nondeterministically "guess" where it is. That's simple enough.

In the PDA I build, you can see the design is followed to a T: in q1, we push items to the stack for every symbol we see, with the added possibility that the machine guesses that this last `a` is the middle character in the string. In the event that this guess is made, we pop continuously until we run out of string, and then if the stack is empty, we accept.

b) Please see the file named `3_pda.png`

c) Please see the file named `3_accept.png`

d) Please see the file named `3_reject.png`

Problem 4:

a)

This PDA is slightly trickier. It's *really* important to keep in mind the corner cases. For example, `222`, `01`, `02`, and {} are all accepted.

The other thing to note is that the number of `0`s is either equal to the number of `1`s or `2`s. The easiest way to make sure we account for this is to count the number of `0`s and compare that number to the number of `1`s and `2`s.

That part isn't hard at all. Note here that these numbers absolutely *must happen in sequence*, e.g., "123", but never "132". This allows us to count the number of `0`s and then the number of `1`s and then the number of `2`s in that order. That way, we don't have to tally up the `0`s and `1`s and `2`s as we go. Much simpler!

So, to do this, we just count the 0s and then branch off and let the machine nondeterministically "guess" whether the number of 0s is equal to the number of 1s or 2s.

So the procedure, at least so far, is to first count the 0s by pushing to the stack every time we encounter one. Then when we encounter some other number, we can just start counting that. To count those numbers, we pop the stack. Then, at the end, if the stack is empty, we accept. Simple.

My implementation is a bit trickier looking that it really is conceptually. I basically implemented the the "core", which does what was described above. The extraneous paths are to account for the corner cases (e.g., things like 222, 01, 02, 022222, 11111, and so on).

b) Please see the file named `4_pda.png`

c) Please see the file named `4_accept.png`

d) Please see the file named `4_reject.png`

Problem 5:

a) This one is actually significantly easier. We only need to track two things: whether the number of 0s is equal to the number of 1s, or is twice that amount. There was some ambiguity here. I chose to interpret the rules to mean that either there are strictly the same number of 0s as there are 1s, or there are twice as many 0s. A minor modification could me made to the PDA to make it so that the number of 1s could be the integer quotient of 0s divided by 2. But this seemed odd too me, so I didn't implement it.

Since PDAs are nondeterministic, I chose to force the automaton to guess at the very beginning. One branch causes the PDA to verify that there are the same number of 0s as there are 1s. This is accomplished by pushing to the stack every time we read a 0 and the popping when we read a 1.

The other branch is only slightly more complicated; we run through two 0s and push only once. Then every time we encounter a 1, we pop. Empty stack is accepted.

It's also useful to note that empty set is included.

b) Please see the file named `5_pda.png`

c) Please see the file named `5_accept.png`

d) Please see the file named `5_reject.png`