Assignment 9

Alex Clemmer
CS 3100
Student number: u0458675

Problem 1:

This minimization is pretty simple:

```
2 .         2 x         2 x         2 x
3 . .       3 x .       3 x x       3 x x
4 . . .     4 x . .     4 x x .     4 x x .
  1 2 3       1 2 3       1 2 3       1 2 3
```

Thus states 3 and 4 get merged.
[INSERT DIAGRAM HERE]

Problem 2:

Problem 3:

Given some FA, say we replace every symbol in its alphabet with $\{a\}$. This in all cases except the most simple DFAs gives us a *strictly* nondeterministic FA (since it is impossible to have many choices going to different states using the same letter). This resulting NFA gives us strings that are the same *length* as the strings generated by the original NFA.

Now, the interesting part. If we transform our new NFA into a DFA, the machine for all non-trivial cases ends up coiling into itself (and the trivial cases will extend our conclusion, as we will see shortly). So why is this?

On a very high level, remember that this NFA is essentially measuring the lengths of the strings of some regular expression. The impact of this is that the lengths of strings ends up being the product of some set of fixed length (sub-)expressions. In other words, in contrast with context-free languages, which can be recursively variable-length (*e.g.*, $\{0^n 1^n\}$, whose second half length is defined by the length of the first half, as opposed to, say, $\{(ab)^*\}$, which must always be a multiple of 2 in length).

The one example of this is the NFAs that reduce to DFAs that do not loop. This *only* happens in trivial cases (*e.g.*, when there is only one state in the DFA. But this turns out to be periodic too, because the length is always a function of the period of 1. So really these are the same case.

Problem 4:

DFAs and BDDs are

Problem 5:

This is not exactly an XOR, but it's close. It currently fails for input `(a=1, b=0)` or `(a=0, b=1)`. Basically what's happening is that