**CS 81, Logic and Computability**
**Problem Set 5: DFAs**

## Problem 1: Deterministic Finite Automata Warmup [25 Points]

As we saw in class, deterministic finite automata (DFA) can be nicely encoded in Prolog! The provided file `DFAaccepts.pl` simulates any DFA assuming that we specify the DFA (typically in another file) as follows:

- The start state is understood to be called `q0`.
- For every accepting state (e.g., `q42`) we include a predicate `accepting(q42)`.
- For every transition from one state (e.g `q42`) to a state (e.g. `q47`) on a given symbol (e.g. 1), we include a predicate of the form `transition(q42, 1, q47)`.

The `accepts` predicate is intended for use with *any* DFA. The code is provided in the `DFAaccepts.pl` file posted with this assignment. Next, we can determine if a *particular* DFA accepts its input this way: First, we load in `DFAaccepts.pl` (e.g., using `consult("DFAaccepts.pl")`) and then, similarly, load in the specific DFA that we've designed. Let's imagine that the DFA is in a file called `dfa1.pl` and it accepts exactly those binary strings in which the number of 1's is even. We've provided such a `dfa1.pl` example file as well. Note that if the machine is to accept any input, this file should contain at least one `accepting` predicate (indicating an accepting state) and one `transition` predicate for every state and every possible input. In general, if the set of states is $Q$ and the alphabet is $\Sigma$, the number of transitions will be exactly $|Q| \times |\Sigma|$. Here's an example of the code in action.

```
?- consult("DFAaccepts.pl").
?- consult("dfa1.pl").

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

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

Notice that the input is represented as a list rather than a string. The list is read left-to-right and the DFA accepts if and only if it is in an accepting state after consuming the last symbol.

Your task in this problem is to write a DFA for the language for alphabet $\{0, 1\}$ such that the number of 0's is a multiple of 2 and the number of 1's is a multiple of 3. Notice that 0 is a multiple of both 2 and 3. Submit this in a file called `dfa1.pl`. Here's what it will look like when you run it in Prolog:

```
?- consult("DFAaccepts.pl").
true

?- consult("dfa1.pl").
true

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

**Problem 2: Binary Addition with DFAs [25 Points]**

Professor Aadi Shawn is studying the following language which we'll call "the language of valid additions." To begin, let our alphabet $\Sigma$ be the set of all $3 \times 1$ binary vectors. There are eight symbols in this alphabet, each of which is a vector.

$$\Sigma = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

A correct addition of two binary numbers can be represented by a string in $\Sigma^*$. For example ...

$$\begin{array}{rcccc} & 0 & 1 & 0 & 1 \\ + & 0 & 1 & 1 & 0 \\ \hline & 1 & 0 & 1 & 1 \end{array}$$

... would be represented by the following string of four symbols from $\Sigma$.

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The language $L$ is the set of all strings in $\Sigma^*$ representing correct additions. Note that the input is read left-to-right, meaning that we see the digits on the highest powers of 2 before digits on the lowest powers of 2.

Your task is to build a DFA for this language in a file called `dfa2.pl` Each of the eight symbols is represented as an array of length 3. For example, the string of four symbols above would be represented as ...

```
[[0, 0, 1], [1, 1, 0], [0, 1, 1], [1, 0, 1]]
```

... and here's what we'd see when we run this:

```
?- consult("DFAaccepts.pl").
true
?- consult("dfa2.pl").
true

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

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

**Problem 3: Multiples of Three in Binary! [25 Points]**

Construct a DFA (in Prolog, of course) that accepts exactly those binary strings that represent multiples of 3 in binary. The input is read left-to-right, which means that we see the most significant digit first. Submit your DFA as `dfa3.pl`. Here are some examples:

```
?- accepts(q0, [1, 1, 0]).  <-- that's the number 6 in binary
true

?- accepts(q0, [0, 0, 1, 1, 1]).  <-- that's the number 7 in binary
false
```

Notice from the second example above that leading 0's are permitted. For full credit, do this in just three states!