1. Give the recursive definition of the following languages. For both of these you should concisely explain why your solution is correct.

   (a) A language L_A that contains all palindrome strings using some arbitrary alphabet Σ.

   Solution: A string $w \in \Sigma^*$ is a palindrome if and only if:

   - $w = \varepsilon$, or
   - $w = a$ for some symbol $a \in \Sigma$, or
   - $w = axa$ for some symbol $a \in \Sigma$ and some palindrome $x \in \Sigma^*$

   ■

   (b) A language L_B that does not contain either three 0's or three 1's in a row. E.g., $001101 \in L_B$ but 10001 is not in L_B.

   Solution: We are going to define two languages, L_{B1} and L_{B0} using a mutually recursive definition. L_{B1} contains all strings in L_B that start with 1, and L_{B0} contains all strings in L_B that start with 0. We are also going to have $\varepsilon \in L_{B0}$ and $\varepsilon \in L_{B1}$.

   Definition:

   - $\varepsilon \in L_{B0}$
   - $\varepsilon \in L_{B1}$
   - If $x \in L_{B0}$, then $1x$ and $11x$ are in L_{B1}
   - If $x \in L_{B1}$, then $0x$ and $00x$ are in L_{B0}

   Then $L_B = L_{B1} \cup L_{B0}$ ■

2. For each of the following problems:

   i. Formulate the problem as a *regular* language (give an example of the problem instances and how they are encoded, you don't have to write every problem instance).

   ii. Describe the regular expression that describes the expression

   Note that how you encode the language matters for the regular expression you end up with.

   a Checking whether (or not) a number is divisible by 4). You are given a binary number and need to output if this number is divisible by 4.

Solution: Part(i) Intuition: Note that if a binary number is divisible by 4, then it must have 2 zeroes in the suffix.

Strategy: Assume we want to formulate this as the language: ($L_{Div4?}$). For every binary number x, we:

- Add the string $w = x \cdot$"|1" to $L_{Div4?}$ if the two-character suffix of $x[0:1] = 00$. (x is divisible by 4). We add the "|" to separate the input from the output of the problem.
- Add the string $w = x \cdot$"|0" to $L_{Div4?}$ if the two-character suffix of $x[0:1] =$ 01 or 10 or 11. (x is *not* divisible by 4).

Part(ii) Formulating the language like we did above makes the regular expression very easy:

$$r_{DIV4?} = \underbrace{(0+1)^*00|1}_{\text{all output 1 instances}} + \overbrace{((0+1)^*(01+10+11)|0"}^{\text{all output 0 instances}}$$

Alternatively, formulating a regular expression that accepts only the binary strings divisible by 4 is also acceptable. Therefore, any equivalents of $(0+1)^*00$ is a valid solution.

■

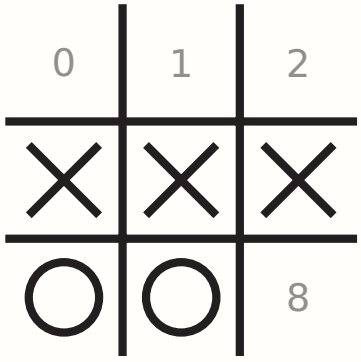
b The game of TicTacToe. You are given a completed tic-tac-toe board and you need to determine who won. (this won't have a clean regular expression. Just define some encoding and describe how you would build the expression, you don't need to write the whole expression out.) Hint: think about how many games of TicTacToe there are.

Solution: Part (i):

Intuition: So with the game of TicTacToe, first thing to notice is that there are a finite number of games. This means that the language that recognizes a TicTacToe board and calculates it's winner must **have a finite number of strings making it automatically regular.**

Strategy: Let's first focus on formulating the strings in the language. First notice that a TicTacToe board has nine spaces and each space can have one of three values (blank, X, O). We can construct an encoding where each square corresponds to a position on the string like follows:

- The language is composed of 9-character strings. Every charcater on the string corresponds to a space on the TicTacToe board.
- We consider a alphabet $\Sigma = \{\square, \times, \circ\}$ to represent a empty-space, X-mark, O-mark.
- Next we concatenate a "|" symbol to note the end of the board encoding and add a character to mark the winner.



This game of TicTacToe would be encoded as
$w = [\square, \square, \square, \times, \times, \times, \circ, \circ, \square, |, \times]$

There are at most 3^9 possible boards (but remember not all boards are valid since the number o X's and O's must differ by at most 1. **We construct the language (L_{TTT}) by including a string for all the possible games of TicTacToe.**

Part(ii)

I'm not going to type out the regular language explicitly but let's define the regular expression as:

$$r_{TTT} = r_{TTT} + w \qquad \forall\ w \in L_{TTT}$$

As mentioned earlier $|L_{TTT}| < 3^9$ which is finite so our above construction is valid.

■

3. **Regular expressions I:** For each of the following languages over the alphabet $\{0,1\}$, give a regular expression that describes that language, and briefly argue why your expression is correct.

   (a) $L_A = \{w \mid |w| \leq 5\}$

   Solution: Anytime you see a length requirement your first instinct is to somehow use a numerical exponent. So maybe something like:

   $$(0+1)^5$$

   But wait! That'll give us all the binary strings of length exactly equal to 5. We could make a expression like :

   $$(0+1)^0 + (0+1)^1 + \ldots + (0+1)^5$$

   or we can be slightly more clever and say:

   $$(\varepsilon + 0 + 1)^5$$

   Both are equivalent, though one is preferable if you don't have access to copy/paste. ■

   (b) $L_B = \{w \mid w \text{ is any string not in } 0^* + 1^*\}$

   Solution: So the language represented by $0^* + 1^*$ contains all the binary strings that have only 0's and all the binary strings that have only 1's. Hence, all the strings not included in that language have at least one 0 and one 1. Something like this would do:

   $$(0+1)^* 0 (0+1)^* 1 (0+1)^* + (0+1)^* 1 (0+1)^* 0 (0+1)^*$$

   Note that we need to account for 01 and 10 so that's why we have to have two parts to the above expression. ■

(c) $L_C = \{w | w \text{ is any string not in } (01^+)^*\}$

Solution: Couple things to unpack here. First, if the string begins with a 1, then regardless of the suffix, that string is not in $(01^+)^*$. So far we got:

$$1(0+1)$$

Now, what do we do about the strings that start with 0's? If we look at the expression $(01^+)^*$, we see that every 0 has a run of 1's between them. Therefore any string with two or more zeros next to eachother is not in the language:

$$1(0+1)^* + (0+1)^* 00 (0+1)^*$$

but we're not done. Again looking at $(01^+)^*$, we see one more condition. Even if we have single 0's separated by 1's, if the string ends with a 0, it's still not in that language. Therefore we need to add:

$$1(0+1)^* + (0+1)^* 00 (0+1)^* + (0+1)^* 0$$

And with that I think we got all the strings that don't appear in $(01^+)^*$. ■

(d) $L_D = \{w | w \text{ every odd position is a } 1\}$

Solution: Let's assume there are a even number of characters. Then the expressions is:

$$\big((0+1)1\big)^*$$

Next what if there's a odd number of strings? Means that there's a extra character that can be either a 0 or 1. Let's add that in:

$$\big((0+1)1\big)^* (0+1)$$

So we just need to combine the two to get the final solution:

$$\big((0+1)1\big)^* + \big((0+1)1\big)^* (0+1)$$

or

$$\big((0+1)1\big)^* (\varepsilon + 0 + 1)$$

And that's it. Simple but effective. ■

4. **Regular expressions II:** For each of the regular expressions, give a brief (1-2 sentence) English description of the language that regular expression represents.

   (a) $\left((0^*10^*10^*)^*\right)^2 \qquad \Sigma = \{0,1\}$

   Solution: Language that contains all strings where the number of 1's is divisible by 2 (not 4!). Think about it this way, the above equation can be re-written as:

   $$\left((0^*10^*10^*)^*\right)\left((0^*10^*10^*)^*\right)$$

   which represents the sets:

   $$\{\varepsilon, 00, 0000, 0101, 0000010100, \ldots\} \cdot \{\varepsilon, 00, 0000, 0101, 0000010100, \ldots\}$$

   Hopefully, it's easy to see now why the number of 0's is divisible by two but not necessarily four. ■

   (b) $\emptyset(0+1)^*1 \qquad \Sigma = \{0,1\}$

   Solution: Empty language. Any set concatenated with the empty set is also empty. ■

   (c) $(\varepsilon+1)(01)^*(\varepsilon+0) \qquad \Sigma = \{0,1\}$

   Solution: All strings with alternating 0's and 1's. We discussed this in lecture. ■

   (d) $1^*|1+(0+1)^*0(0+1)^*|0 \qquad \Sigma = \{0,1,|\}$

   Solution: Represents the problem of bitwise AND operation on n-bits. The value to the right of the "|" symbol is the logical AND of the bits on the left of the "|". Again, a big thing I want to emphasize is that languages are used to represent problems and that there is a point to what you're learning. ■