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 decision problem[1] P:

   i. Describe a way to represent the inputs as strings(the alphabet doesn't have to be binary).

   ii. Describe a regular expression that represents the language $L := \{w \mid P(w) = 1\}$ and briefly explain why the regular expression is correct.

   Note : You don't have to provide the exact regular expression for part (ii). Describing how you would formulate the regular expression is sufficient, given that the exact regular expression could indeed be obtained by following your solution.

   a Input : A binary number
   Output : *TRUE* if the input is divisible by 4, *FALSE* otherwise

Solution:

- **Part(i)**
  Since the input is a binary number, the input as is could be a string representation with binary alphabet.
- **Part(ii)**
  Note that if a binary number is divisible by 4, then it must either be 0 or have 0s for the two least significant digits. Assume we want to formulate this as the language: ($L_{Div4?}$). Then, $L_{Div4?}$ is a set of all strings that ends with two consecutive 0s, unioned with {0}. Therefore the following is the regular expression for $L_{Div4?}$.

$$r_{DIV4?} = (0+1)^*00 + 0$$

■

[1] A decision problem is a mapping from the set of all inputs to the boolean set {*TRUE*, *FALSE*}. That is, any yes-or-no problem is a decision problem, and vice versa.

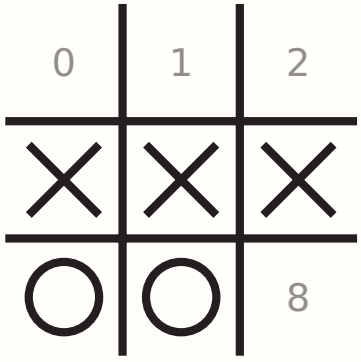
b Input : A complete and valid tic-tac-toe board filled with O and X
Output : *TRUE* if the winner is O, *FALSE* otherwise

Solution: Part (i):

Intuition: The first thing to notice is that there are a finite number of valid board configurations. This means that the language that recognizes a TicTacToe board on which O is the 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.



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

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, and also there can only be one winner). **We construct the language (L_{TTT}) by including a string for each valid game of TicTacToe where O is the winner.**

Part(ii) We can construct the regular expression by taking the union of every element in L_{TTT}. As mentioned earlier $|L_{TTT}| < 3^9$ which is finite so our 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) All strings that end in 1011.

   Solution: $(0+1)^*1011$.

   The expression $(0+1)^*$ matches a string of 1's and 0's of arbitrary length. The substring 1011 in the end of the regular expression ensures all matching strings must end with it. ■

   (b) All strings except 11.

   Solution: $\epsilon + 1 + 0 + 00 + 01 + 10 + (0+1)^3(0+1)^*$

   The first part $\epsilon + 1 + 0 + 00 + 01 + 10$ covers all strings up to length 2, explicitly not listing 11. The expression $(0+1)^3(0+1)^*$ represents all strings of length three or greater. ■

   (c) All strings that contain 101 or 010 as a substring.

   Solution: $(0+1)^*(101+010)(0+1)^*$.

   The expression $(0+1)^*$ refers to all binary strings of length zero or more. A string with 101 or 010 substring, would have 101 or 010 with any number of pre/post-ceding 0's and 1's. ■

   (d) All strings that contain 111 and 000 as a subsequence (the resulting expression is long – describe how you got your expression, instead of writing it out explicitly).

   Solution: Let S be the set of all strings of length 6 that have the desired property. That is, we have

$$S = \left\{ \begin{array}{l} 000111, 001011, 001101, 001110, 010011, \\ 010101, 010110, 011001, 011010, 011100, \\ 111000, 110100, 110010, 110001, 101100, \\ 101010, 101001, 100110, 100101, 100011 \end{array} \right\}.$$

   There are $\binom{6}{3} = 6 \cdot 5 \cdot 4/6 = 20$, such strings, and they all belong to our language. For a string s, let $f(s)$ be the regular expression of inserting $(0+1)^*$ between any two characters of s, and also in the beginning of s and the end of s. For example, we have $f(01) = (0+1)^*0(0+1)^*1(0+1)^*$. The desired expression is

$$\sum_{s \in S} f(s).$$

   ■

(e) The language containing all strings that do not contain 111 as a substring.

Solution: $((\varepsilon + 1 + 11)0^+)^*(\varepsilon + 1 + 11)0^* = ((\varepsilon + 1 + 11)0)^*(\varepsilon + 1 + 11)0^*$.

A *run* is a string s is a maximal substring of s, all made of the same character. Given a string w in the language, break it into blocks, where a block starts just before a run of 1s start. For example,

$$010001101011 = 0|1000|110|10|11.$$

Such a block starts with a run of 1s of length zero, one, or two. It is then followed by a run of at least one 0. The only exception is the last block, which does have to end with zeros. ■

4. **Regular expressions II:** 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) All strings that do *not* contain 000 as a subsequence.

   Solution: This is just a complicated way to say that the string can contain at most two 0s. Before, in between and after the zeros, we can have a run of ones. As such, the solution is: $1^*(\varepsilon + 0)1^*(\varepsilon + 0)1^*$. ■

   (b) Strings in which every occurrence of the substring 00 appears before every occurrence of the substring 11.

   Solution: $(10 + 0)^*(1 + 10)^*$.

   Indeed, break the string in the language into two parts. The prefix p that contains no consecutive 1s, and the suffix s that contain no consecutive 0s.

   The expression $(10 + 0)^*$ includes all strings with no consecutive 1s, and matches p. Similarly, the expression $(1 + 10)^*$ includes all strings with no consecutive 0s. ■

   (c) Strings that do not contain the subsequence 010.

   Solution: $1^*0^*1^*$.

   The condition implies that there could not be two runs of zeros in the string. ■

   (d) Strings that do not contain the subsequence 0101010.

   Solution: $1^*0^*1^*0^*1^*0^*1^*$.

   The condition implies that there could be at most three runs of zeros in the string. ■

   (e) Strings that do not contain the subsequence 10.

   Solution: 0^*1^*.

   Such strings can not contain a run of 1s followed by a run of 0s. As such, it can contain only a run of 0s followed by a run of 1s. ■