1 Problems as Languages

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).

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

1. 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: Homework 1 Problem. ■

2. The sum of two *unary* integers.

   Solution: Let's first refresh on what unary numbers are. A unary number is base 1. In other words, each digit in a unary number represent $1^{digit_position}$. Example: $\mathbf{5}_{10} = \mathbf{101}_2 = \mathbf{11111}_1$

   So the language would be similar to the example discussed in lecture. Let's assume $\Sigma = \{1, +, =\}$. We don't particularly need the "+" and "=" symbols but I added them for visual clarity. Therefore we can describe the language as:

   $$L_{+unary} = \left\{ \begin{array}{ccc} +=, & +1=1, & +11=11,\ldots \\ 1+=1, & 1+1=11, & 1+11=111,\ldots \\ 11+=1, & 11+1=111, & 11+11=1111,\ldots \\ \vdots & \vdots & \vdots \\ n+=1....1(\text{n times}), & \ldots, & n+11=1....1(\text{n+2 times}),\ldots \end{array} \right\} \quad (1)$$

   ■

3. The game of TicTacToe. You are given a completed tic-tac-toe board and you need to determine who won. Hint: think about how many games of TicTacToe there are.

   Solution: Homework 1 Problem. ■

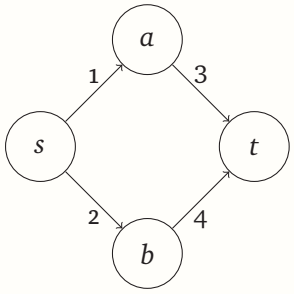
4. The shortest path of a weighted (only positive integer values), undirected graph between 2 nodes s and t.

Solution: Again, this is just a issue of embedding a graph as a string which can be accomplished a multitude of ways. Big thing is to remember what a graph is, a set of nodes and a set of edges. So how do we encode a graph as a string? We simply list out the nodes and edges.

Each string in a language represents an *instance* of the problem. In this case, the problem input is the weighted, undirected, graph and the output is the shortest path. Knowing this, we can formulate a string embedding such as:

$$\langle < listofnodes > | < listofedges > | < shortestpath > \rangle$$

Hence if we have a graph such as:



would yield a sting embedding like:

$$\langle \underbrace{a,b,c,d}_{\text{Nodes in Graph}} | \underbrace{s1a,s2b,a3t,b4t}_{\text{Edges in Graph}} | \underbrace{sat}_{\text{Shortest Path}} \rangle \tag{2}$$

Again I'm including a bunch of extaneous notation for the sake of visual clarity. So the language that defines the shortest path problem would simply be a collection of all these strings.

■

2 Recursive Definitions

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

1. A language that contains all strings.

   Solution: So the thing to remember with recursive definitions is that you have to have a base case, and a set of rules that could build on so let's do that:

   Base case: $\varepsilon \in L_a$

   Next we got the inductive step where we define rules to make other strings. The good news is that since this language is any strings, we can simply say:

   - $w = \mathtt{0}x$ for some $x \in L_a$, is in L_a
   - $w = \mathtt{1}x$ for some $x \in L_a$, is in L_a

   ■

2. A language which holds all the strings containing the substring **000**.

   Solution: Very similar to the last problem expect we need to ensure all the strings in L_b have a particular substring. We can accomplish this by simply changing the base case:

   - **Base case:** $\mathtt{000} \in L_a$
   - $w = \mathtt{0}x$ for some $x \in L_a$, is in L_a
   - $w = \mathtt{1}x$ for some $x \in L_a$, is in L_a

   But this isn't completely correct because it means the will always be a suffix. Easy to fix with a few more rules:

   - $w = x\mathtt{0}$ for some $x \in L_a$, is in L_a
   - $w = x\mathtt{1}$ for some $x \in L_a$, is in L_a

   ■

3. A language L_A that contains all palindrome strings using some arbitrary alphabet Σ.

   Solution: Homework 1 Problem. ■

4. A language L_B that does not contain either three **0**'s or three **1**'s in a row. E.g., $\mathtt{001101} \in L_B$ but $\mathtt{10001}$ is not in L_B.

   Solution: Homework 1 Problem. ■