Language Recognition

- The unifying framework into which any computational problem can be cast is language recognition.
- Given:
  - The definition of language L
  - A string w.

  we ask the question: "Is w in L?". This question is a part of a more general class of prblem called the decision problem which requires a yes or a no answer.

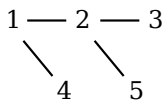
Notes

The Framework

- The Languages and strings concept give us a mathematical model to work with that can be used to solve computation problems of all sorts.
- The language recognition task is similar to the way of defining a set using a *characteristic funtion* that decides whether an element is a part of the set or not.
- "Is w in L?" is the decision that we have to take. That becomes our main task. The Language will obviously have to be defined and all the strings have to examined to make sure it's in L.

Encoding

- Now we have the fundamental question to ask, "Is w in L?" which might seem limiting. So we need to figure out a way to "recast" a computational problem, more complex into the form "Is w in L?"
- We use *encoding* for that. We express the objects as strings using an encoding and resulting strings are a part of the language L. We can then ask the question "Is w in L?"
- **Everything is a string**. Using strings as an abstraction is useful for the mathematical model we are building. Dealing with strings is what is crucial in theory of computation. And in fact the most basic and fundamental units of computers are 0s and 1s which are essential the two characters in the alphabet and every combination of them is a string. And all of computing is formed by these strings of 0s and 1s.
- The encoding will obviously have to be specified for each problem. For example A graph can be encoded as follows
  - Start with the binary representation of the number of vertices $|V|$
  - List the edges in the graph as pairs of vertices represented by a pair of binary numbers
  - separate the binary numbers using '/'



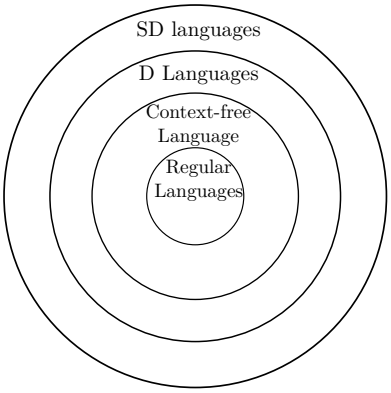
The above graph can then be represented as:
`101/1/10/10/11/1/100/10/101`

Thus we have encoded a graph as a string. If G was the graph then $< G >$ represents the string encoding of the graph.

- Encoding simply lets us express problems' entities as strings and we can then define a language and ask the fundamental question "Is w in L?"
- For problems that are not decision problems we can reformulate them using encoding to turn them into decision problems. The idea is to have the inputs and the outputs of a problem P into a single string. This way we can have a language constructed that has the correct outputs corresponding to the correct inputs for that problem.
- When we have the decision question we can reduce it to the original problem. This simply means that if we have a program that solves P we can use it to construct a program that decides the language L for it. And if we have the language L we can construct the program to solve P by checking if the encoded problem string (inputs and outputs in a single string) for the particular instance of the problem, is in L or not.

Machine-based hierarchy of language classes

- Regular languages (FSM)
- Context-Free languages (PDA)
- Decidable and Semidecidable languages (Turing Machine)



- **Regular Languages** - languages that can be accepted by some FSM are called regular languages. We can have an FSM for

$$L = \{w \in \{\texttt{a},\texttt{b}\}^* : \text{all a's come before b's in } w\}$$

- **Context-Free Languages** - languages that can be accepted by a PDA. A PDA is a pushdown automata and is similar to an FSM but it also has a stack. the notation $x/y/z$ is used as a label for an arc to show that if a is read then pop y from the stack if possible and push z to the stack if possible and move to the pointed state. $x/\epsilon/z$ would mean "don't check the stack but push z if x read". $x/y/\epsilon$ would mean pop y and don't push anything if x read. We can build a PDA for:

$$L = \{a^n b^n : n \geq 0\}$$

- **Decidable Languages** - languages for which a Turing machine halts on all inputs. It's guaranteed to give a 'yes' or a 'no'. One such example is:

$$L = \{a^n b^n c^n : n \geq 0\}$$

- **Semidecidable Languages** - languages for which a Turing machine is not guaranteed to halt. It may accept a string in L but may not be able to reject a string not in L i.e., it might not know when to stop looking for an answer.