Week One, Lecture One

About this Course

Imagine a Venn diagram with the left set comprised of "discrete math," "words," and "analysis" and the right set of "continuous math," "pictures," and "synthesis." Most math courses are strictly one set, usually the left. This course is in the intersection of the two sets: between discrete and continuous math; mathematical rigor and creativity.

This course has practical implications in compiler construction.

"What can a computer do?"

The central question of this course. Some answers from the class were:

- storage of information
- perform mathematical instructions
- computation
- automate tasks

The professor's list was:

1. Inputs
2. Outputs
3. Programmable
4. Storage
5. Initial state
6. Deterministic
7. Finite number of operations at a time
8. Problem the computer is solving
9. Classes of problems

In the above list, 8 and 9 are not strictly the computer itself but its environment.

Models

Every suspension bridge in the world is a parabola. Before we had a model for them, we didn't necessarily realize this but gravitated towards this conclusion naturally.

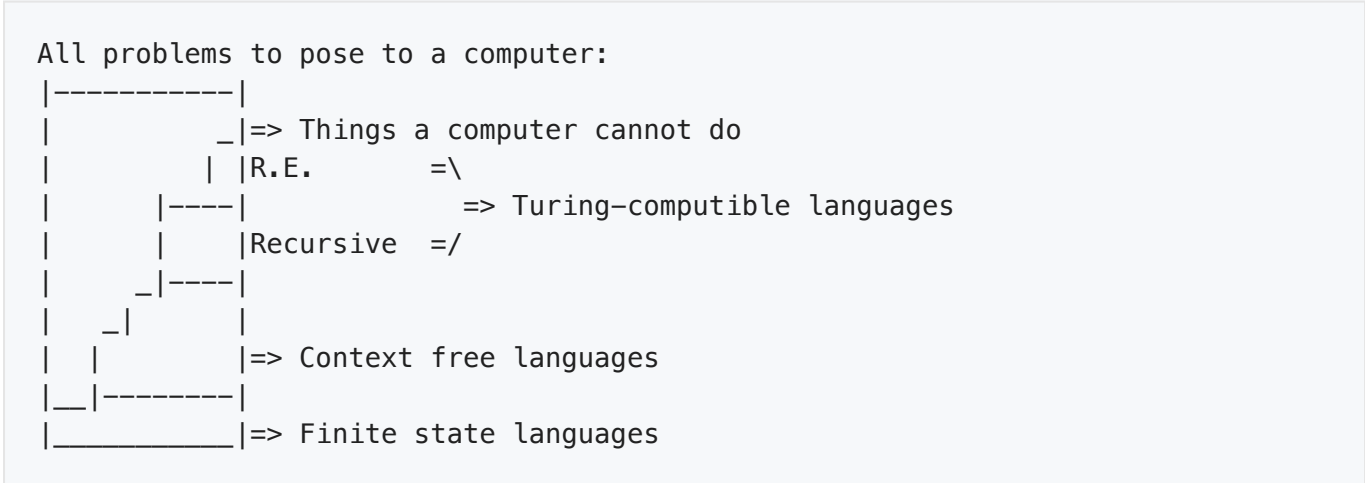
We can analyze the suspension bridge according to different aspects, called **models**. Analogously, in linguistics we analyze language through *syntax*, *semantics*, *phonology*, etc. These are all different models of language.

A football field has specifications (models) such that - despite the complexity of the field's configuration - you only need to specify two numbers: its length and its width. Once you know

these two numbers, you use the model of a football field and get a universally recognized result.

Finite Automaton

ex: A graph which contains phrases and directed edges to other phrases from the lyrics of "Modern Love" by David Bowie. This represents an FA (finite automaton).



FSL are a subset of CFL, which are a subset of TCL. As the diagram progresses, you increase "power." (e.g. TCL are more powerful than CFL).

- FSL represent keywords of a programming language
- CFL represent *syntax* of programming language (not semantics)
- TCL represent anything you can do with a computer
    - RE stands for "recursively enumerable"

Also note that as the diagram progresses, we have the left part of the rectangle decrease. This decreasing part represents "understandability" - the idea that, as our computers grow more and more powerful, we as humans are less able to understand how they arrived at their conclusions.

Three ways to represent a computer as a mathematical object:

1. FSA (Finite state automata) - in line with FSL
    - Least powerful
2. PDA (Pushdown automata) - in line with CFL
3. Turing machine - in line with TCL
    - Most powerful

Languages and Computability

We think of computers as doing arithmetic. But we can also think of it as a relationship between the operands e.g. rather than thinking of addition as a process (add(2,3) = 5), we can think of addition as a relationship of three numbers represented as a triple ((2,3,5)). Instead of thinking

of the computer taking in 2 and 3 and generating 5 (*computation problem*), we think of the computer as taking in (2,3,5) and generating `true` (*recognition problem*). Further, is human language syntax a computation problem or a recognition problem or both?

Baby's First Model of Computation

Imagine a box with a lightbulb sticking out of it. We connect this lightbulb-box to a tape. This tape has multiple cells, each with a symbol. For each symbol, it does some processing. At each cell, the lightbulb will either be on or off. If it is on, the input so far is *acceptable*. If it is off, the input so far is *unacceptable*.

- ex: our tape reads `|2|+|3|=|5|` and by the time we read 5, our light is on

Referring to our earlier list, this machine can be labelled as follows:

1. Input = tape
2. Output = lightbulb
3. Program = Box
4. Storage = Box
5. Initial state = Box
6. Deterministic = We read the tape from left to right and our internal graph (discussed below) has each node with only one edge per input

Our State Representation

Inside the box, we have a graph. Each symbol read from each cell of the input tape moves our graph to a certain node on the graph. Each node is a **state** of the machine. Certain nodes are defined as **accepting states**. If we reach the end of the tape and are in an accepting state, then the lightbulb is on.

Example

Our alphabet = `{0,1}` Our initial state is a node with just one incoming edge Our input = all strings with an even number of 1's Our FSA is represented as:

`e` is an *accepting state*, so if we receive an input with an even number of 1's we will be in an accepting state.