WEEK 1, LECTURE 2 ON 21 AUGUST 2021 CS1.301.M21 ALGORITHM ANALYSIS AND DESIGN

Recall

The main questions asked regarding computational problems:

- What is a *computational problem*?
- How to prove correctness? Is it efficient?
- How to prove optimality?
- Can we do better?

All problems are membership queries in languages.

COUNTABILITY AND COMPUTABILITY

It turns out that there are uncountably many computational problems but there are only countably many programs to solve them. Hence there are uncountably many unsolvable problems.

When working with C programs or Turing machines there is no loss of generality. So the number of programs is the same as the number of C programs.

A countable set is one which has a bijection $f : \mathbb{N} \to S$ from the natural numbers to S. i.e. if every element in a set can be sensibly mapped to an index in an array, then we can say it is countable.

Theorem: {1,0}* is countable

ε	0	1	00	01	10	11	000	001	010	011	100...
1	2	3	4	5	6	7	8	9	10	11	12...

Any finite length string of 0s and 1s can be mapped to a natural number.

-Hence the number of possible programs is countably infinite.

Theorem: P({0,1}*) is uncountable

A problem was defined as a (membership query in a) language. So the total possible number of problems is $P(\{0,1\}^*)$

Any subset of $P(\{0,1\}^*)$ can be viewed as a unique $f : l\{0,1\}^* \rightarrow \{0,1\}$ which is 0 when an element does not belong to the selected subset and 1 when it does. Like shown:

x	ε	0	1	00	01	10	11	000	001	010	011	100	101	111	0000	...
$f(x)$	1	1	1	1	0	0	1	1	0	1	0	0	1	1	1	...

Hence these languages can be represented by such strings.

Listing out all such languages:

	ε	0	1	00	01	10	11	...
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4	1	1	0	1	0	0	0	
L_5	1	1	1	1	1	1	1	
L_6	0	0	1	0	0	0	1	
...								

And taking all the bits along a diagonal, flipping them and listing them as a language will always give one that is yet to be listed.

So once all natural number in $\mathbb{N}$ are assigned a language, there is always another one that is yet to be assigned.

- **Therefore, $P(\{0,1\}^*)$ is uncountable**.

WITH REGARDS TO COMPUTABILITY

The number of programs is countable and an uncountable number of problems.

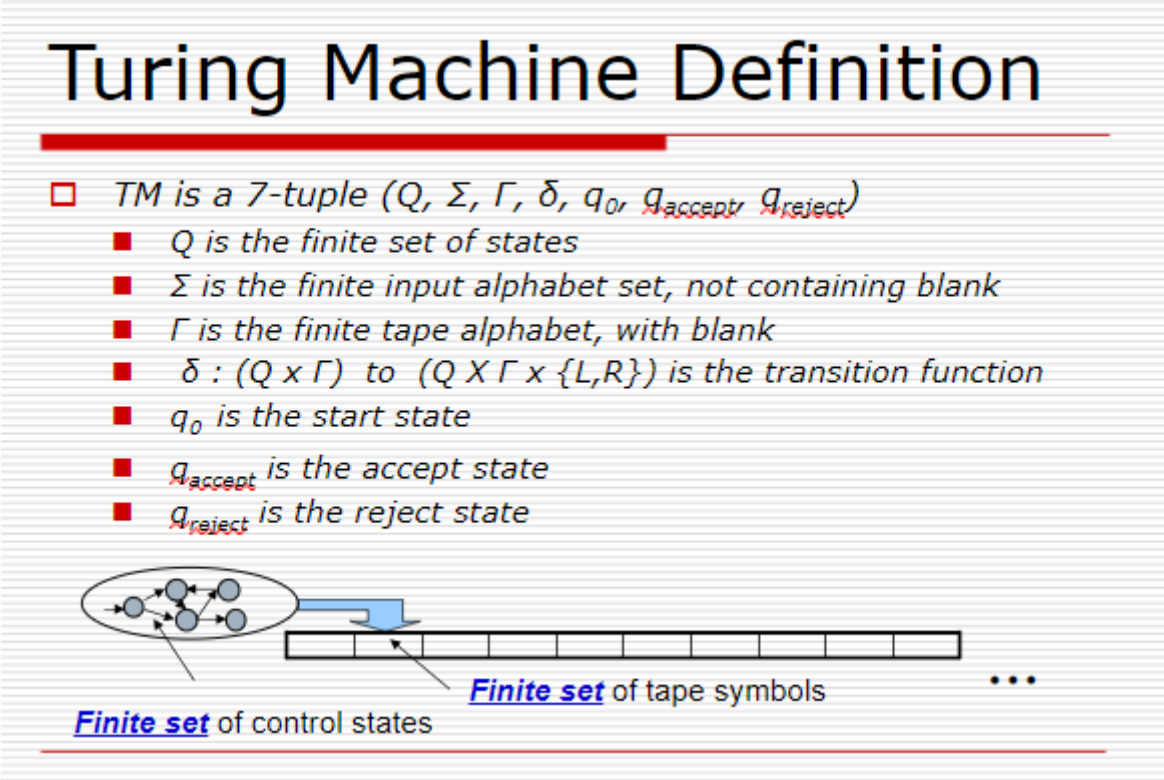
Hence the number of problem exceeds the number of possible solutions and proves the existence of non-computable problems.

Eg. *Program equivalency problem*: Write a program that checks whether two other programs given as input solves the same problem.

> As long as the axioms of computing stand, no futuristic computer will also be able to compute these non-computable problems.

The Church-Turing Hypothesis

A function is only computable if it is computable by a Turing machine:



It is known that C programming is Turing complete. Hence the non existence of a C program also implies non-computability.

In that case, when discussing computability we can also work with pseudo code.