CSCI 301 - Lecture 17: Intro to Theory of Computation

Theory of Computation

- computability theory
  - what problems can computers solve?
- complexity theory
  - how "hard" are different classes of problems?
- automata theory
  - toolbox for formalizing the above

Computability Theory Example: The Halting Problem

H: given a program p and an input I, return true if P halts (terminates) on input I.

Examples:

```
P: (+ 2 2)
```

```
Q(i):
  while i>0:
    i+=1
```

- H(P, 0) → true!
- H(Q, 1) → false
- H(Q, 0) → true

Theorem: A program that computes H does not exist

Proof (contradiction): Suppose H exists.

Construct a program Z as follows:

```
Z(String x):
  if H(x,x):
    loop forever
  else:
    return
```

Run: $Z(Z)$

→ Evaluates $H(Z,Z)$

What does Z do if

- $H(Z,Z)$ returns true? loop forever
- $H(Z,Z)$ returns false? terminates

Complexity Theory Example: P vs NP

Subset sum problem:

Given a (multi)set of positive integers, is there a subset that sums to T?

$S = \{2, 3, 7, 8, 10\}$

$T = 11$

Verify a solution: $C = \{3, 8\}$

```
sum = 0
for v in C:
    sum += v
return sum == T
```

Find a solution:

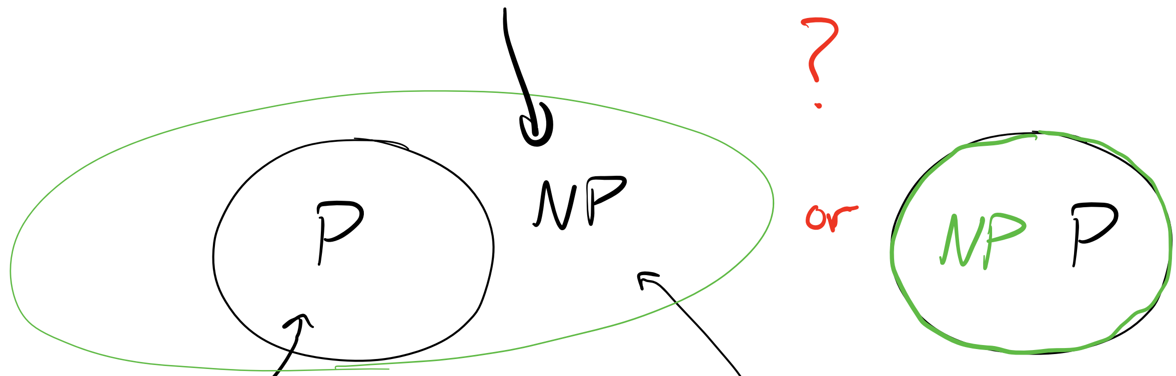
```
for C ∈ P(S):
    if verify(C):
        return true
return false
```

How many loop iterations, in terms of $n = |S|$?

$\leq n$	2^n

verified in "polynomial time"

not solved in "polynomial time"



Set of problems solvable in polynomial time

Set of problems verifiable in polynomial time

To answer "what problems can be solved (efficiently) using a computer, we need to formalize what we mean by computer and problem.

For this purpose, we have

Automata Theory

To formalize "computer", we define abstract machines called automata (plural of automaton)

To formalize "problem", we focus on language acceptance problems

Example automaton: a toll gate state machine.

Toll costs 15¢; gate accepts 5¢ and 10¢ coins.
(n)ickel (d)ime