SFU

Decidability

CMPT 125
Mo Chen
SFU Computing Science
30/3/2020

Lecture 33

Today:

- The Omnipotence of Computers

The Omnipotence of Computers

Time Magazine - April 16, 1984

"Put the right kind of software into a computer, and it will do whatever you want it to. There may be limits on what you can do with the machines themselves, but there are no limits on what you can do with software."

- Editor of a software magazine.

Not so!

Disclaimer

Today's class does not answer the following:

- Can computers run companies?
- Can computers make good decisions?
- Can computers diagnose?
- Can computers love?

While interesting, these are all:

- non-analytic questions
- pseudo-scientific issues

Decision Problems

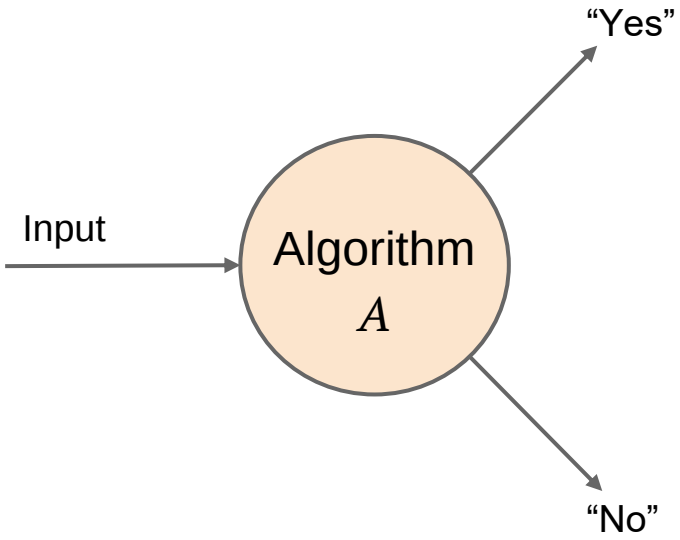
Algorithmic Problem:

- Set of legal inputs
- Specification of desired output as a function of input

like “Accept” / “Reject”

Decision Problem:

- An algorithmic problem where the output is “Yes” or “No”.



Algorithm A:

- involves “effectively executable” elementary operations, each taking bounded time and bounded resources.
- halts for every legal input, and answers the question correctly.

The Nature of the Problem

There are problems that have no algorithm

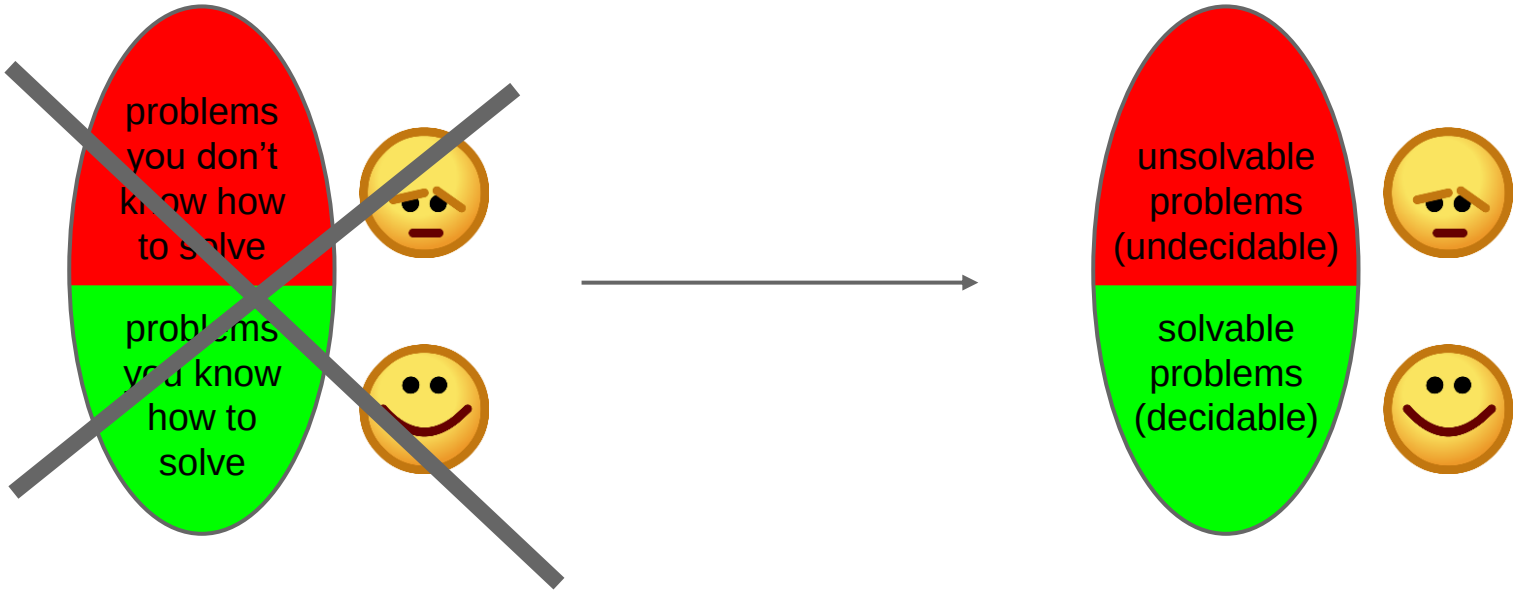
Perhaps the failure is due to insufficient

- money → buy a larger, more sophisticated computer
- time → wait longer for output or use a server farm
- brains → design a cleverer algorithm

But even with unlimited resources, there are some problems that defy solution

- such problems are called *undecidable*

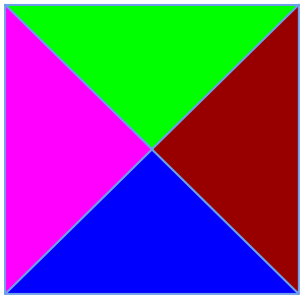
Decidability



Robustness follows from Church-Turing Thesis

- All computers have equivalent algorithmic power.

Tiling Problems



A Tile

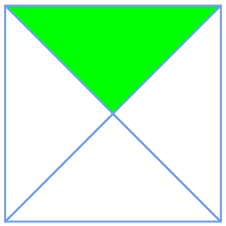
- 1x1
- 4 colours
- Non-rotatable
- Non-reversible

Problem: Tile a portion of the integer grid (perhaps all of it), such that adjacent edges have matching colours.

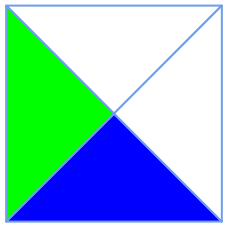
Input: A finite set of tile types, each of which you may use an infinite number of times.

Output: “Yes”, if possible. “No”, if impossible.

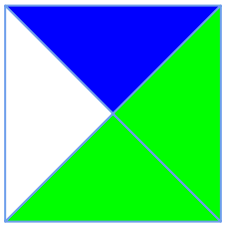
Tiling Puzzle #1



1

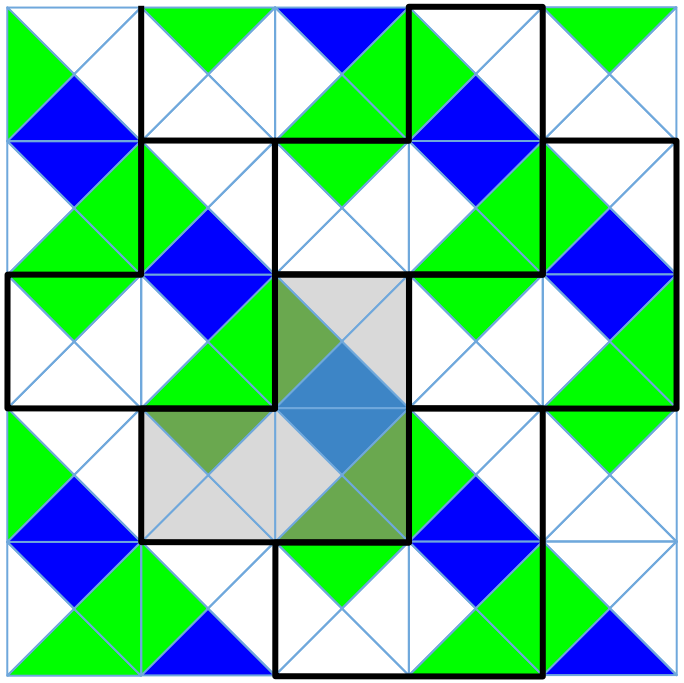


2

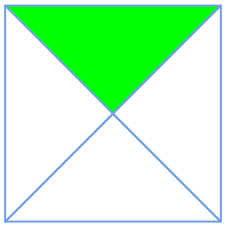


3

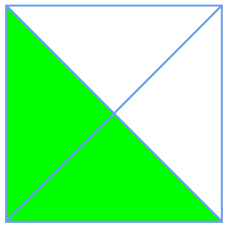
Can tile the entire plane with these!



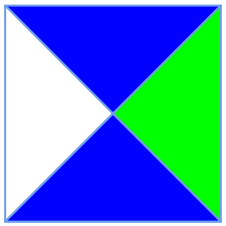
Tiling Puzzle #2



1



2

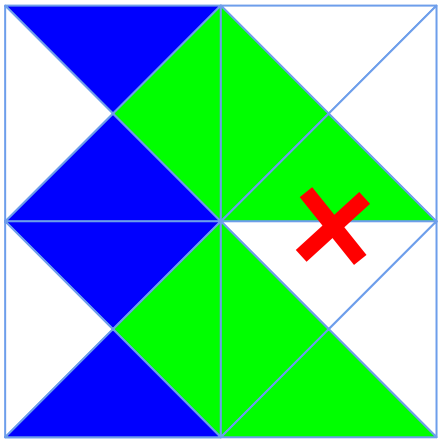


3

Can’t even tile a 3x3 square!

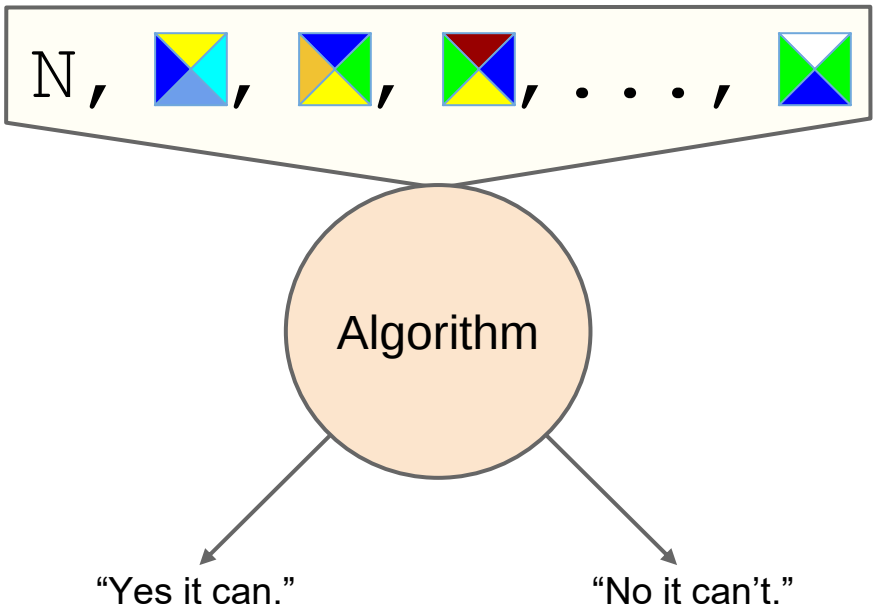
Proof:

- #3 must appear somewhere
- adjacent #3’s need green to the right of it (#2’s)
- Contradiction



Tiling The Plane

Problem: Given a finite set of tiles T, can T be used to cover all of the integer plane?

Input: 

No such algorithm exists!

- No such machine!
- No such program!

Tiling the plane is undecidable!

Undecidable Problems

What other problems are undecidable?

The Halting Problem: Given a program P and an input x, ask if P halts on input x.

- E.g., P: `while(x != 1) { x = x - 2; }`

Naive algorithm to decide halting:

- simulate P on input x
- if P terminates on x then return “Yes”
- else return “No”

Q. What’s wrong with this approach?

Rice's Theorem

Can you write a program to . . .

- find bugs in other programs?
- determine if two programs are equivalent?
- determine whether a program is malicious?
- determine if a program always outputs integers?
- …

virus-checking is undecidable too

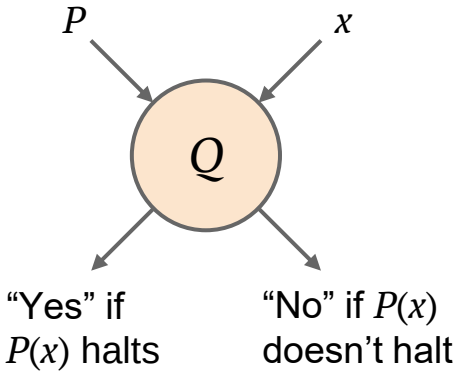
Rice's Theorem: You cannot write a program that determines any non-trivial property about all programs

- can't cover them all, but sometimes can do "most"

Proving Undecidability

There is no algorithm that decides the halting problem for *all possible* program-input pairs

Proof: (Alan Turing - 1936)



- Assume one exists and seek a contradiction!

  That is, there is an algorithm Q that correctly decides whether or not algorithm P halts on x
- Construct an algorithm S that uses Q.
  - Takes w as input
  - Runs $Q(w,w)$
  - if $Q(w,w)$ returns "No", then return "Yes"
  - if $Q(w,w)$ returns "Yes", then infinite loop
- Run $S(S)$. Does it halt?
  - If no, then $S(S)$ returns "Yes"
  - If yes, then $S(S)$ runs forever