Computability and Complexity

COMP 199B 2008

Paris, 1900

- German mathematician David Hilbert presented ten problems in mathematics from a list of 23 (1, 2, 6, 7, 8, 13, 16, 19, 21 and 22).
- Speaking on 8 August 1900, at the Paris 2nd International Congress of Mathematicians, at La Sorbonne. The full list was published later.
- The problems were all unsolved at the time, and several of them turned out to be very influential for 20th century mathematics.

Fundamental question ?

- Can we prove all the mathematical statements that we can formulate ? (Hilbert's 2nd problem)
- Certainly, there are many mathematical problems that we do not know how to solve.
- Is this just because we are not smart enough to find a solution ?
- Or, is there something deeper going on ?

computer science version of these questions

- If my boss / supervisor / teacher formulates a problem to be solved urgently, can I write a program to solve this problem in an efficient manner ???
- Are there some problems that cannot be solved at all ? and, are there problems that cannot be solved efficiently ??
(related to Hilbert's 10^{th} problem)

Kurt Gödel

- In 1931, he proved that any formalization of mathematics contains some statements that cannot be proved or disproved.

Alan Turing

- In 1934, he formalized the notion of **decidability** of a language by a computer.

A Language

- Let Σ be a finite alphabet. (ex: {0,1})
- Let Σ^* be all sequences of elements from this alphabet. (ex: 0, 1, 00000, 0101010101,...)
- A language L is any subset of Σ^*.
- An algorithm decides a language if it answers Yes when x is in L and No otherwise

Alonzo Church

- In 1936, he proved that certain **languages** cannot be **decided** by any algorithm whatsoever...

Emil Post

- In 1946, he gave a very natural example of an **undecidable** language…

(PCP) Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 tiles.

- A solution to **PCP**

aa	bbb	b		
a	a		bb	bb

Post Correspondence Problem

u_1	u_2	u_3	. . .	u_n
v_1	v_2	v_3		v_n

- Given n tiles, u_1/v_1 … u_n/v_n
  where each u_i or v_i is a sequence of letters.

- Is there a k and a sequence $\langle i_1, i_2, i_3, \ldots, i_k \rangle$
  (with each $1 \leq i_j \leq n$) such that

  $$u_{i_1} | u_{i_2} | u_{i_3} | \ldots | u_{i_k} = v_{i_1} | v_{i_2} | v_{i_3} | \ldots | v_{i_k} \, ?$$

A Solution to Post Correspondence Problem

$$\boxed{\begin{matrix} u_1 \\ v_1 \end{matrix}} \quad \boxed{\begin{matrix} u_2 \\ v_2 \end{matrix}} \quad \boxed{\begin{matrix} u_3 \\ v_3 \end{matrix}} \quad \cdots \quad \boxed{\begin{matrix} u_n \\ v_n \end{matrix}}$$

- A solution is of this form:
  with the top and bottom strings identical.

$$\boxed{\begin{matrix} u_{i_1} \\ v_{i_1} \end{matrix}} \quad \boxed{\begin{matrix} u_{i_2} \\ v_{i_2} \end{matrix}} \quad \boxed{\begin{matrix} u_{i_3} \\ v_{i_3} \end{matrix}} \quad \boxed{\begin{matrix} u_{i_4} \\ v_{i_4} \end{matrix}} \quad \boxed{\begin{matrix} u_{i_5} \\ v_{i_5} \end{matrix}} \quad \cdots \quad \boxed{\begin{matrix} u_{i_k} \\ v_{i_k} \end{matrix}}$$

Post Correspondence Problem

- **Theorem:**

  The Post Correspondence Problem cannot be **decided** by any algorithm (or computer program). In particular, no algorithm can identify in a finite amount of time the instances that have a negative outcome. However, if a solution exists, we can find it.

Post Correspondence Problem

- **Proof**:

  Reduction technique - if **PCP** was **decidable** then another **undecidable** problem would be **decidable**.

The Halting Problem

- Notice that an algorithm is a piece of text.
- An algorithm can receive text as input.
- An algorithm can receive an algorithm as input.
- The Halting Problem:
  Given two texts A,B, consider A as an algorithm and B as an input. Will algorithm A halt (as opposed to loop forever) on input B?

The Halting Problem

- **Theorem**: no algorithm can decide the Halting Problem.
- **Proof**: Assume for a contradiction that an algorithm **Halt(A,B)** exists to decide the Halting Problem.

The Halting Problem

- Consider the Algorithm:

  Bug(A)

  if Halt(A,A) then While True do

  { when **Halt(A,A)** is true then **Bug(A)** loops }

  { when **Halt(A,A)** is false then **Bug(A)** halts }

- Question: What is the outcome of **Bug(Bug)**?

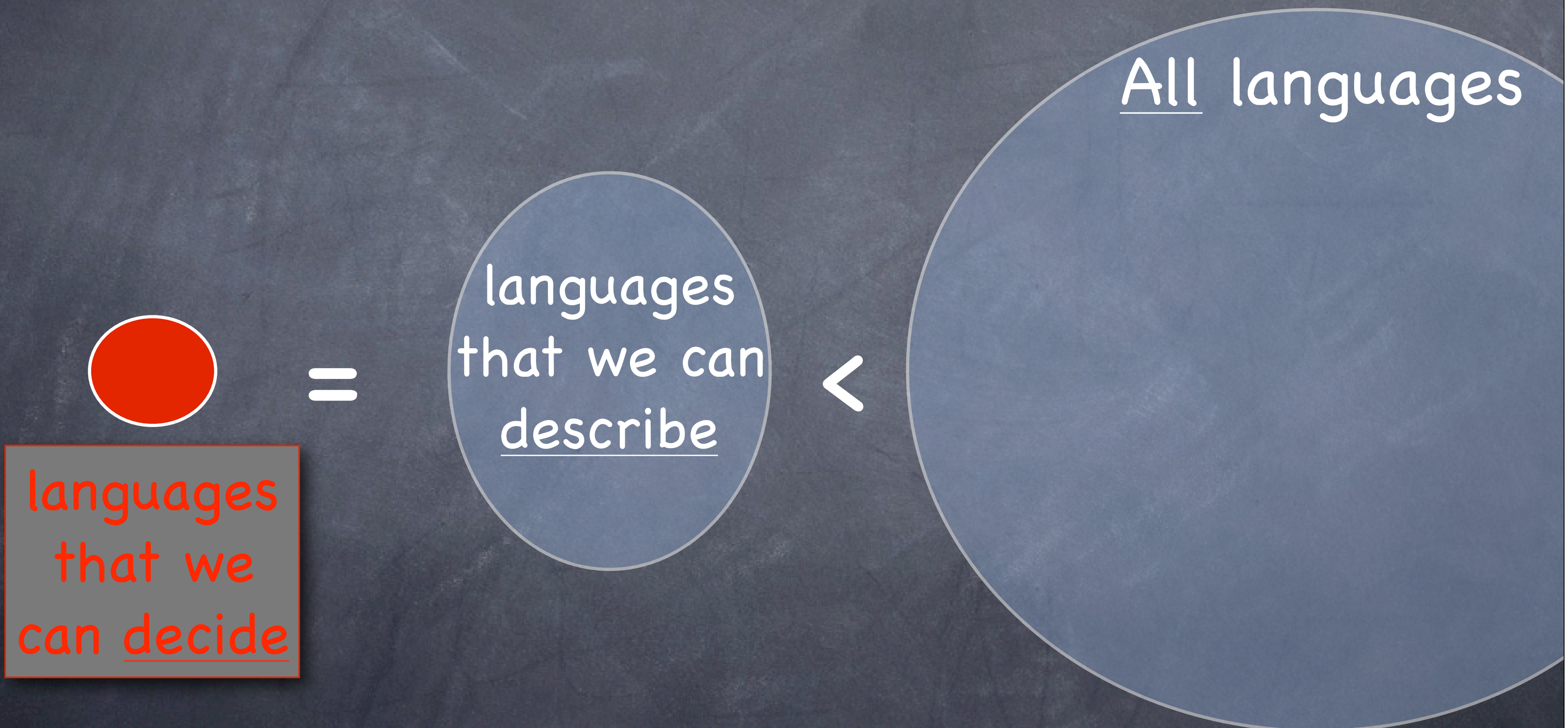
The Halting Problem

- If **Bug(Bug)** does not loop forever it is because **Halt(Bug,Bug)=False** which means **Bug(Bug)** loops forever. (contradiction)
- If **Bug(Bug)** loops forever it is because **Halt(Bug,Bug)=True** which means **Bug(Bug)** does not loop forever. (contradiction)
- Conclusion: **Halt** cannot exist.

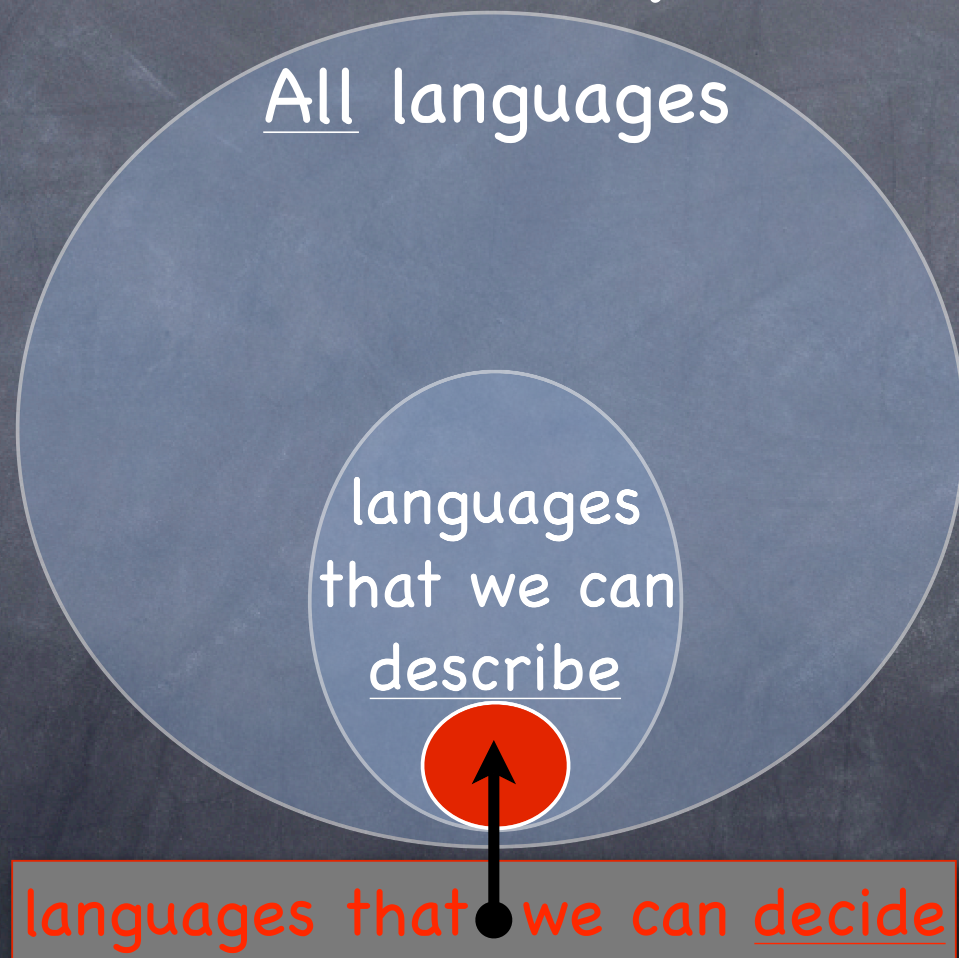
The Halting Problem and **PCP**

- Any algorithm to decide **PCP** can be converted to an algorithm to decide the Halting Problem.
- **Conclusion**: **PCP** cannot be decided either.

Comparing Cardinalities



Computability Theory



Decidable ? Some times we just don't know...

COMP 199B 2008

Syracuse Conjecture

- For any integer n>0 define the following sequence:

$$S_1=n,\ S_{i+1}=\begin{cases} S_i/2 & \text{if } S_i \text{ is even,} \\ 3S_i+1 & \text{if } S_i \text{ is odd.} \end{cases}$$

- **Syracuse**(n)= $\begin{cases} \text{least } i \text{ s.t. } S_1=n,\dots,S_i=1 \\ 0 \text{ if } S_i\neq 1 \text{ for all } i. \end{cases}$

Syracuse Conjecture

- Example: **Syracuse**(9) = 20
- S_1=9, S_2=28, S_3=14, S_4=7, S_5=22, S_6=11, S_7=34, S_8=17, S_9=52, S_{10}=26, S_{11}=13, S_{12}=40, S_{13}=20, S_{14}=10, S_{15}=5, S_{16}=16, S_{17}=8, S_{18}=4, S_{19}=2, S_{20}=1

Syracuse Conjecture

- For all n that we have computed so far, **Syracuse**(n) > 0.
- **<u>Conjecture</u>**

  for all n>0,   Syracuse(n)>0

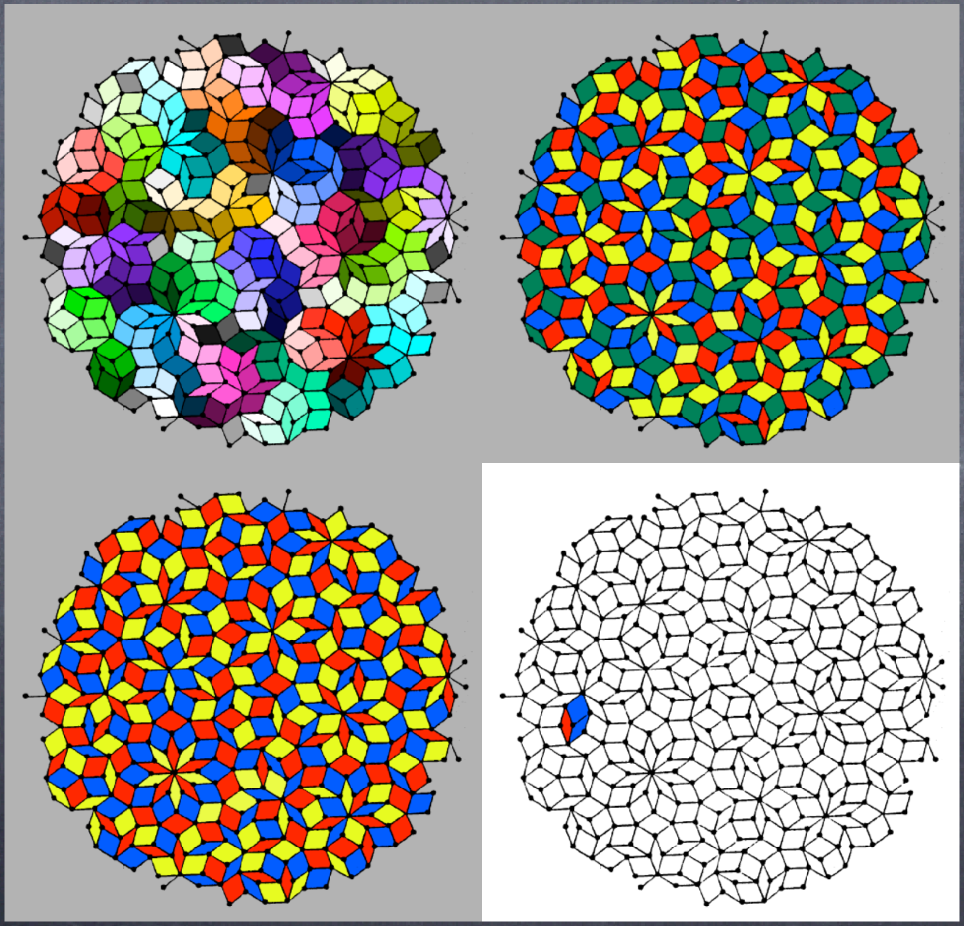
- If there exists N such that **Syracuse**(N) = 0 we might not be able to prove it.

Syracuse Conjecture

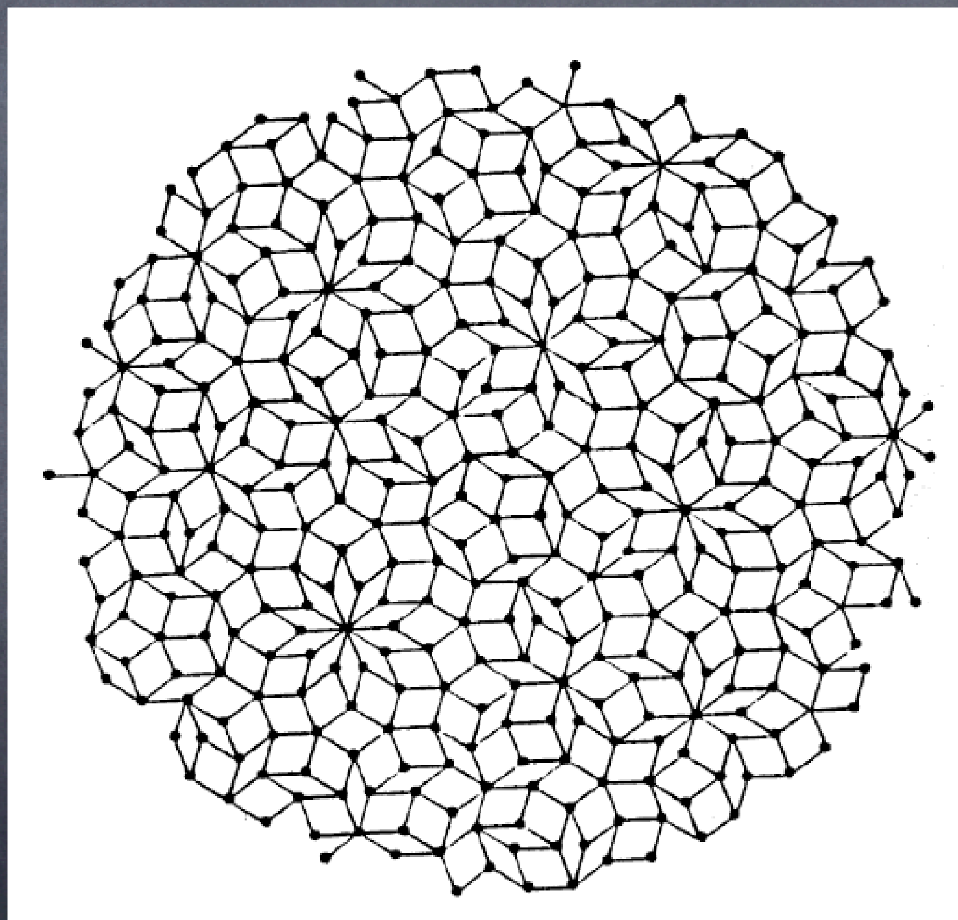
- The Syracuse conjecture is believed to be true but no proof of that statement was discovered so far. It is an **open** problem.
- Even worse, it might be decidable but there might be no proof that it is !!!

Complexity and Tractability

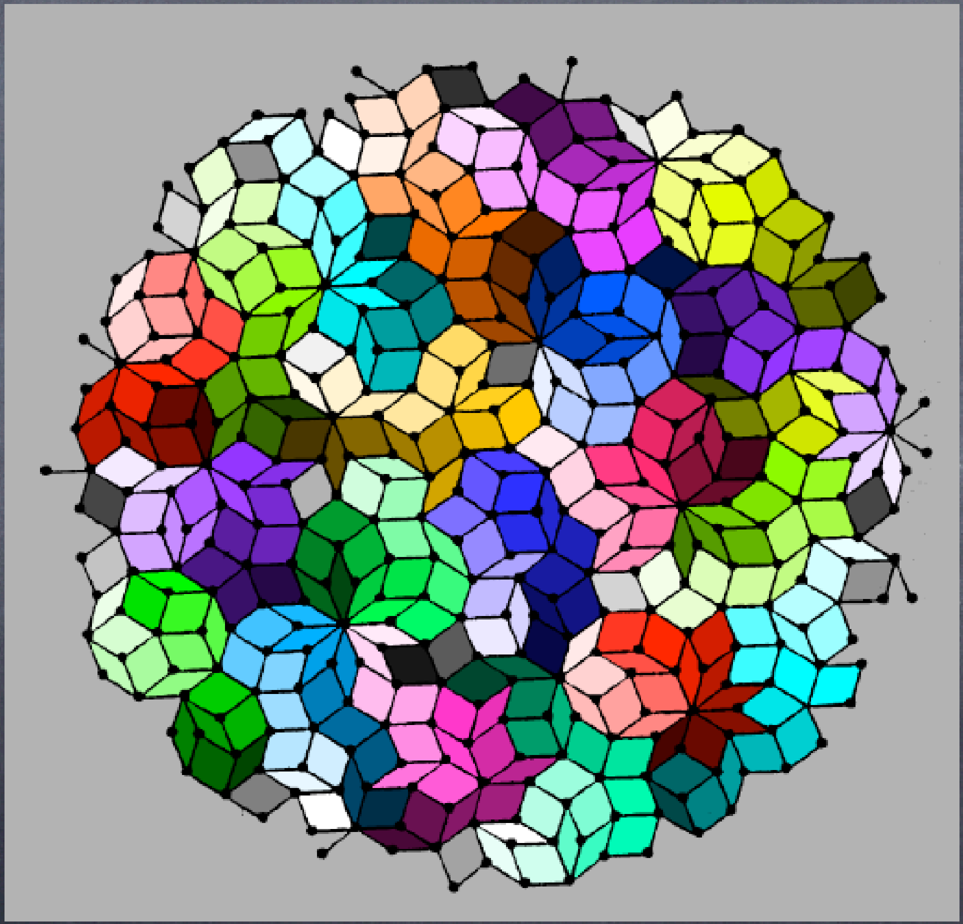
Not all problems were born equal...



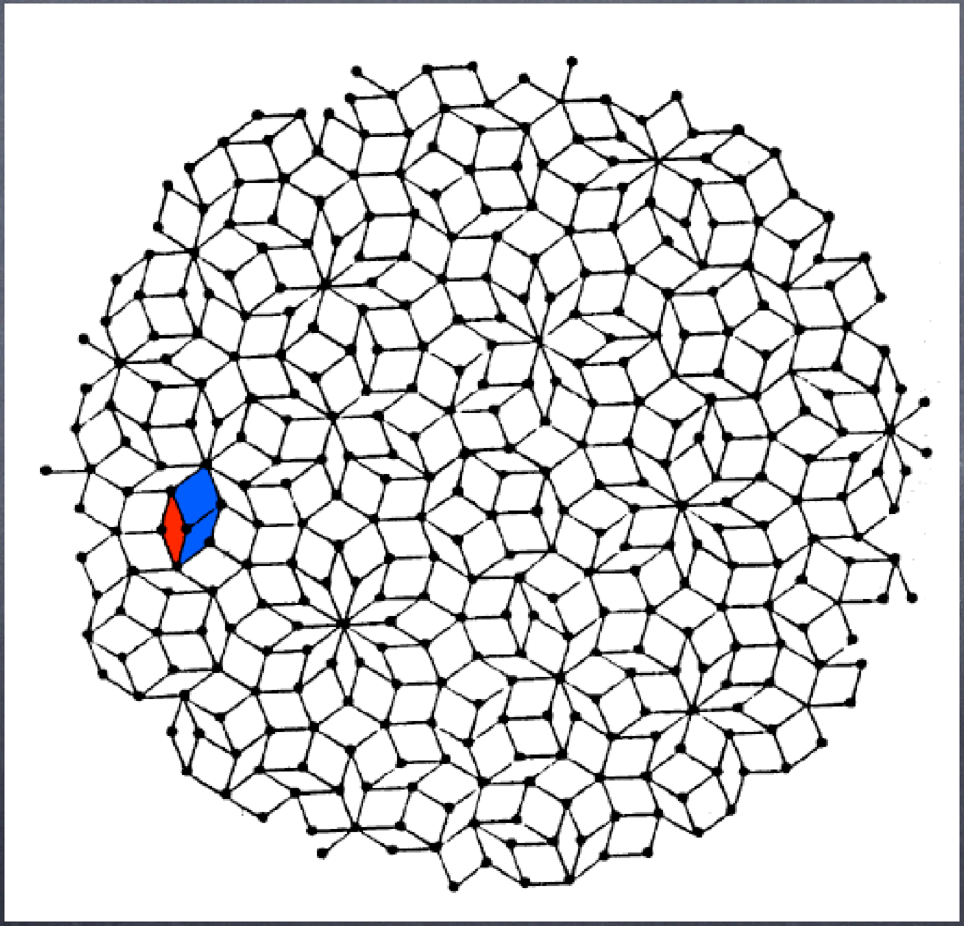
Is it possible to paint a colour on each region of a map so that no neighbours are of the same colour ?



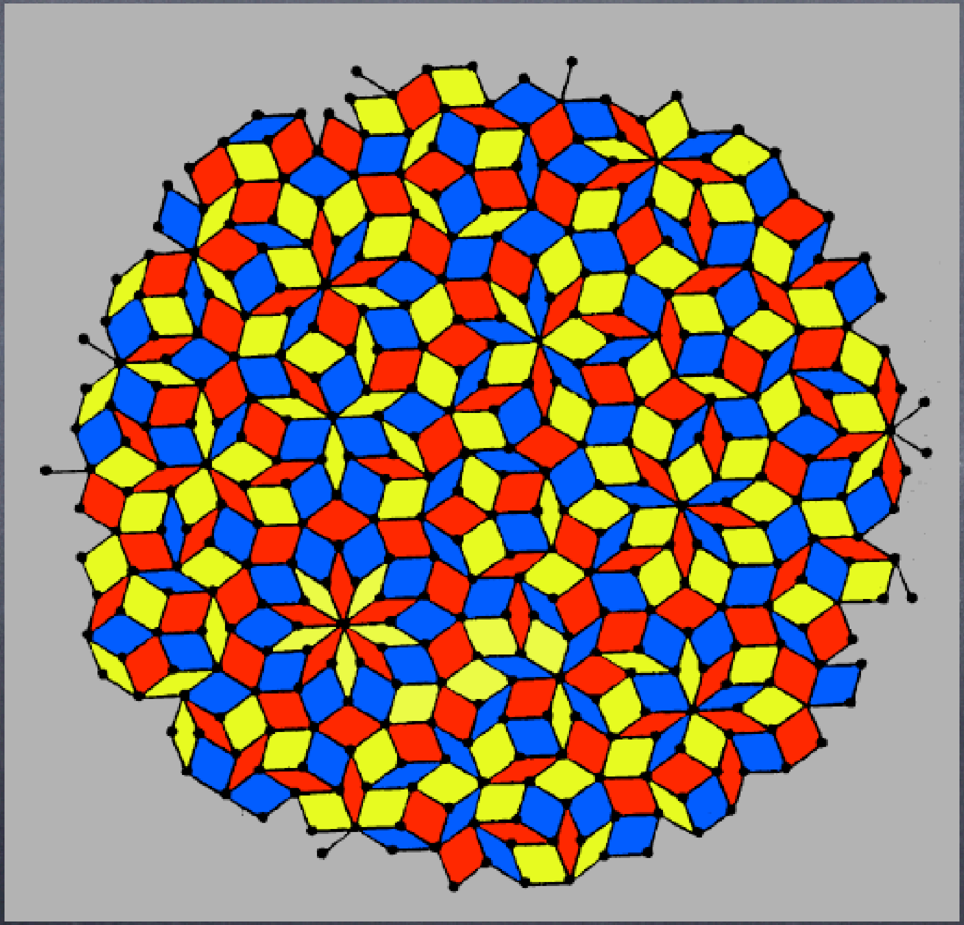
Obviously, yes, if you can use as many colours as you like...



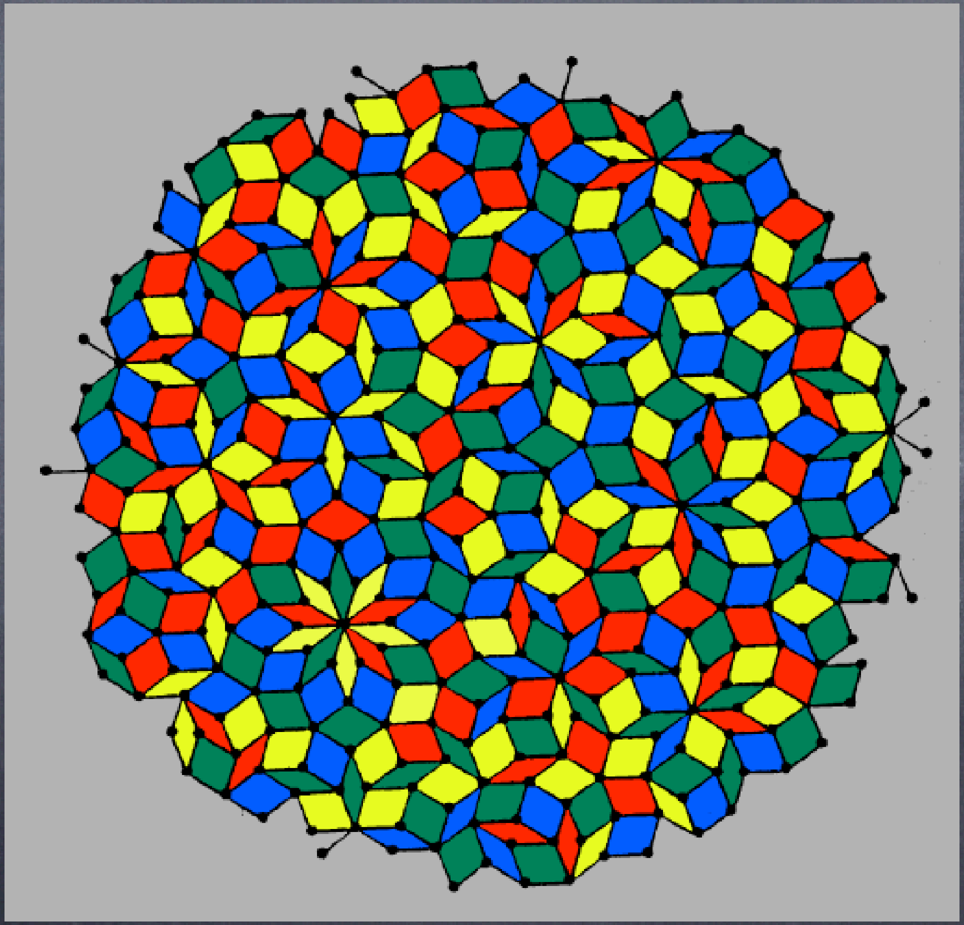
2 colouring problem



3 colouring problem



4 colouring problem



K-colouring of Maps (planar graphs)

- K=1, only the map with zero or one region are 1-colourable.
- K=2, easy to decide. Impossible as soon as 3 regions touch each other.
- K=3, No known efficient algorithm to decide. However it is easy to verify a solution.
- K≥4, all maps are K-colourable. (hard proof) Does not imply easy to find a K-colouring.

3-colouring of Maps

- Seems hard to solve in general,
- Is easy to verify when a solution is given,
- Is a special type of problem (NP-complete) because an efficient solution to it would yield efficient solutions to MANY similar problems !

Examples of NP-Complete Problems

- SAT: given a boolean formula, is there an assignment of the variables making the formula evaluate to true ?
- Travelling Salesman: given a set of cities and distances between them, what is the shortest route to visit each city once.
- KnapSack: given items with various weights, is there of subset of them of total weight K.

NP-Complete Problems

- Many practical problems are NP-complete.
- Some books list hundreds of such problems.
- If any of them is easy, they are all easy.
- In practice, some of them may be solved efficiently in some special cases.

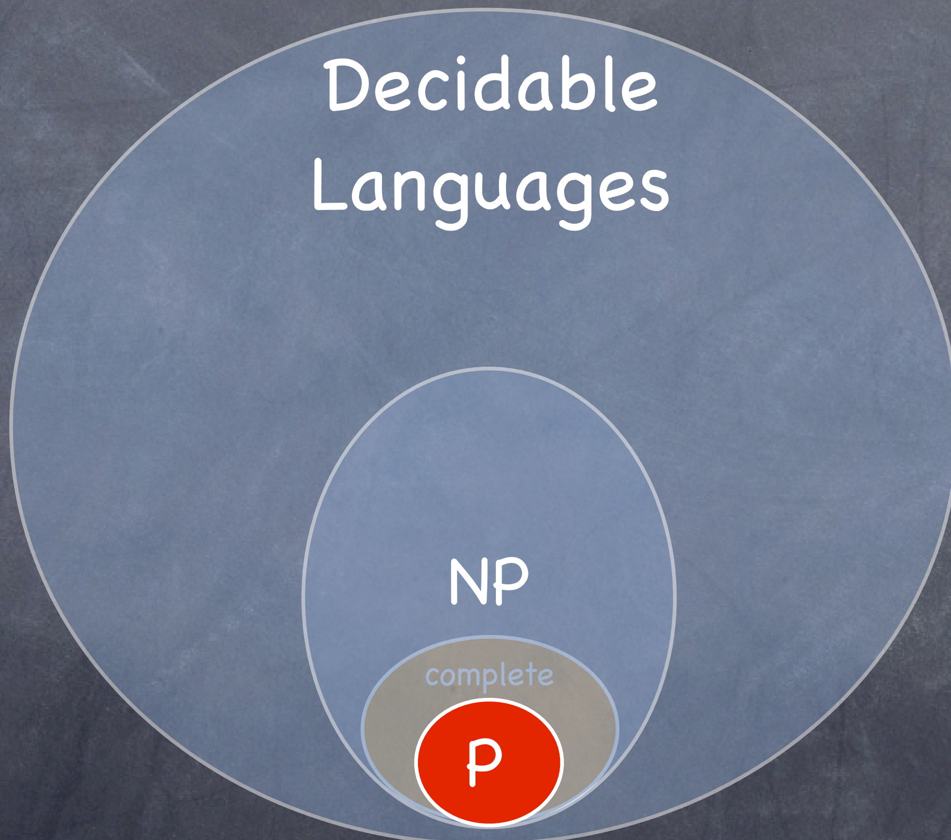
Tractable Problems (P)

- 2-colorability of maps.
- Primality testing.
- Solving NxNxN Rubik's cube.
- Finding a word in a dictionary.
- Sorting elements.

Tractable Problems (P)

- Fortunately, many practical problems are tractable. The name P stands for Polynomial-Time computable.
- Computer Science studies mostly techniques to approach and find efficient solutions to tractable problems.
- Some problems may be efficiently solvable but we might not be able to **prove** that...

Complexity Theory



$P = NP$?

Beyond NP-Completeness

- P-Space Completeness: problems that require a reasonable (Poly) amount of **space** to be solved but may use very long time though.
- Many such problems. If any of them may be solved within reasonable (Poly) amount of time, then all of them can.

P-Space Completeness

- Geography Game:

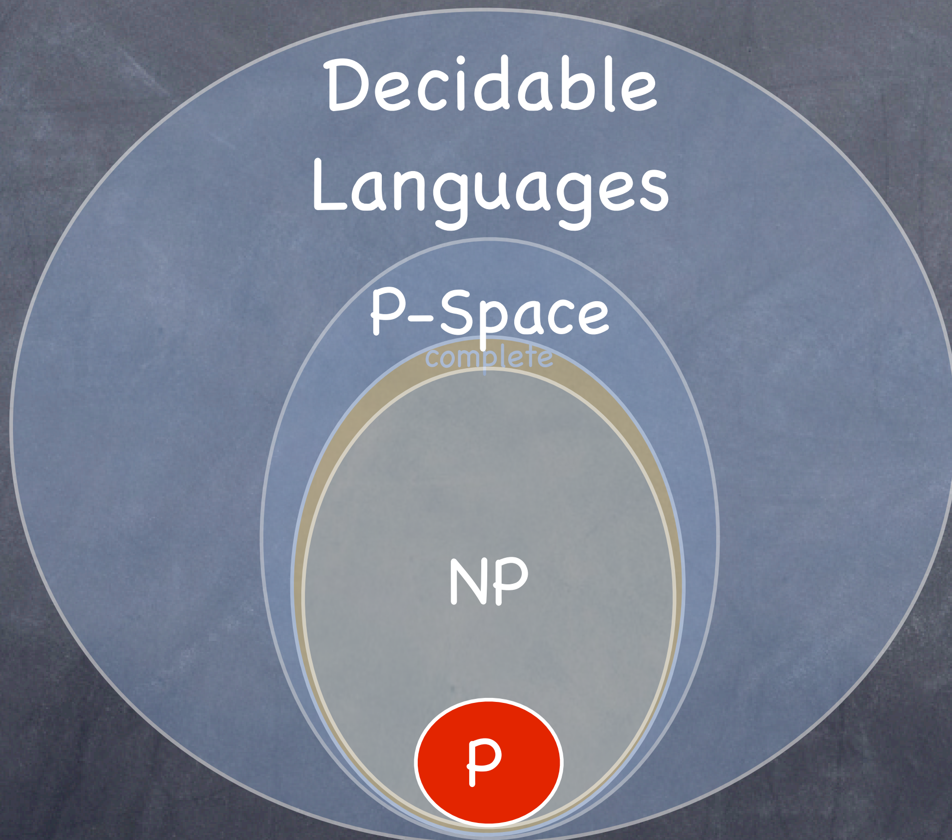
  Given a set of country names: Arabia, Cuba, Canada, France, Italy, Japan, Korea, Vietnam

- A two player game: One player chooses a name. The other player must choose a name that starts with the last letter of the previous name and so on. A player wins when his opponent cannot play any name.

Generalized Geography

- Given an arbitrary set of names: $w_1, \ldots, w_n$.
- Is there a winning strategy for the first player to the previous game ?

Complexity Theory



NP = P-Space ?

Theoretical Computer Science

- **<u>Challenges of TCS:</u>**
- **FIND** efficient solutions to many problems.
- **PROVE** that certain problems are **NOT** computable within a certain time or space. (With applications to cryptography)
- Consider new models of computation. (Such as a Quantum Computer)