Computability and Computation

Chris Lomont

Jun 22,2012

What is computability?

- Once thought any problem, if well specified, could be solved by some deterministic method, i.e., an algorithm.
- Led to study of the class of problems solvable by a machine.
- Note this includes all “algorithms”, such as QuickSort, Euclid’s GCD, H.264, etc.
- Every “computable” function has an algorithm
  - Finite set of statements, each precise, with some rules
  - Must work for any size arguments

Leibnitz dream

- ***characteristica universalis***
- Wanted a universal scientific language
- Mathematicians (there were no computer scientists back then!) believed that we would eventually discover tools that we could answer any question we wanted
  - provided we could express it in the language of logic
- (He also refined the binary number system in the late 1600's)

Hilbert Problems

- List of 23 open problems posed in 1900
  - by David Hilbert
- Drove much of mathematical research in the 20th century
- Examples
  - 2nd: Prove the axioms of arithmetic consistent
    - Led to computation, Turing Machines, ultimately Gödel Incompleteness
  - 6th: Axiomatize all of physics
  - 8th: Riemann Hypothesis (most important open problem in all of mathematics)
  - 18th: What is the densest sphere packing?

Computable Sets

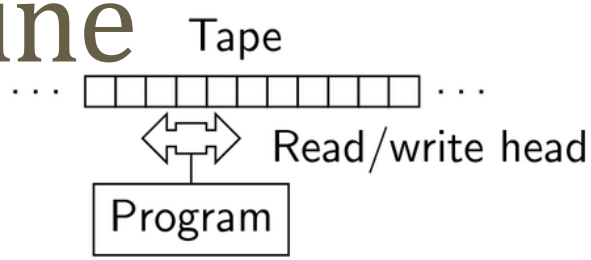
- A set A of natural numbers is computable (recursive, decidable) if there is a computable function f st $f(n) = 1$ if n is in A else $f(n) = 0$.
- Examples
  - Empty set
  - All numbers
  - All finite sets
  - All primes

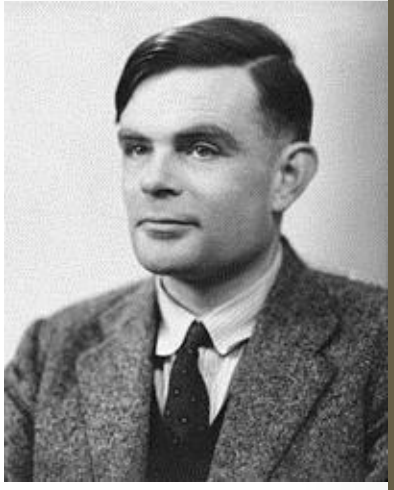
- Most subsets of the natural numbers are *not* computable.
  - All subsets has cardinality “uncountable infinity” while the class of programs is “countable infinity”.

Basic Machine Steps

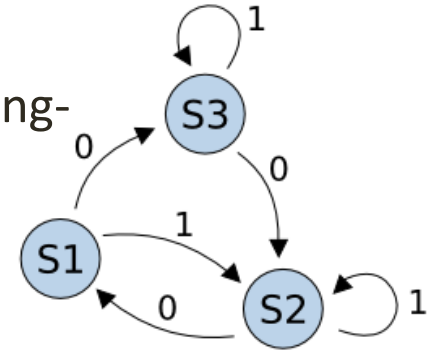
- Originally thought of as steps doable by b=pencil and paper, i.e., by a “computer”
- Local
  - Each step can change at most a finite number of things.
- Transforms one symbol to another
- Example: long division

Turing Machine



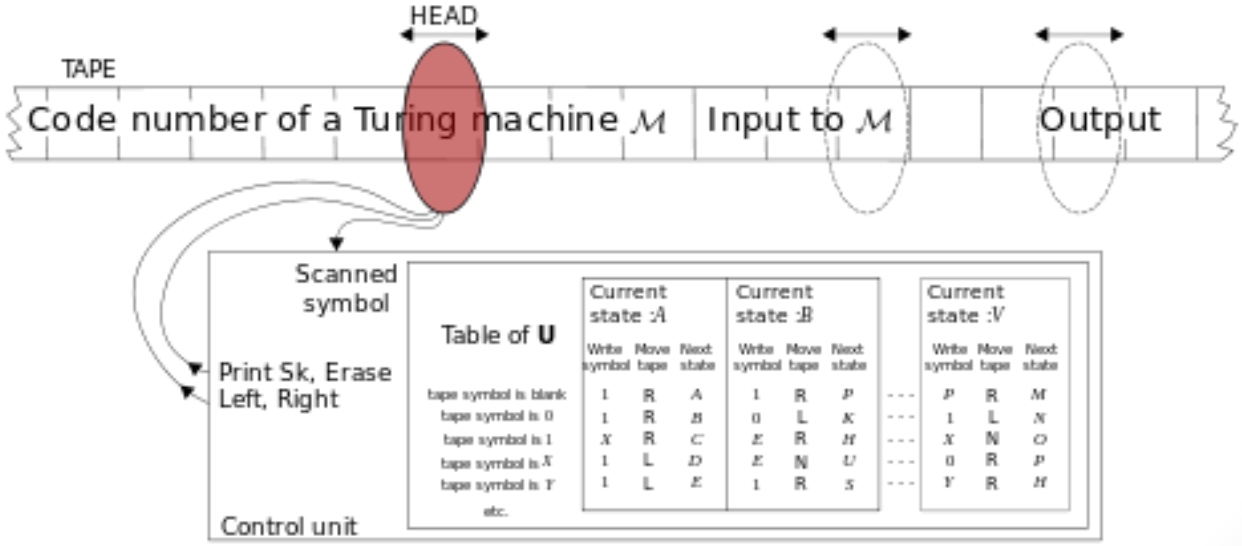


- Alan Turing, 1936
- Infinitely long tape for work space
- Finite program, can be represented as Finite State Machine.
- *computable numbers:* a number is Turing-computable if there exists a Turing machine which starting from a blank tape computes an arbitrarily precise approximation to that number.
  - All of the algebraic numbers (roots of polynomials with algebraic coefficients) and many transcendental mathematical constants, such as *e* and π are Turing-computable.



Universal Turing Machine

- A single machine that can emulate any TM
- UTM "Program" is a copy of the TM to emulate and the program to feed the TM
- Requires at most a logarithmic speed slowdown for the multi-tape version.



Other machine types

- Multi-tape
- Register Machine
- Random Access Machine (RAM)
- Multidimensional (1D tape ->2D or 3D or higher space)

- *All equivalent in the class of problems they can solve*

- Probabilistic (chooses among multiple transitions randomly)
  - Slightly different?

Halting problem

- Can there be a machine M that given any program P and input N decides if P(N) halts or runs forever?
- No! (Turing, 1936)
- The Halting problem is *undecidable* in a TM.
- Gives an *undecidable* problem which is the basis now for many undecidable problems. Places limits on “knowledge”
- Note in many cases automated program analysis can decide.

Proof of Halting Problem

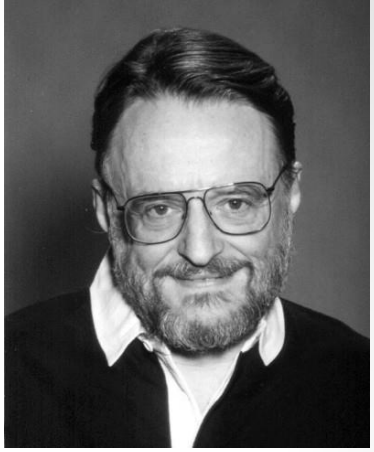
- *Assume* M exists, decides the Halting problem,
  - Enumerate all programs on some TM, then we can treat a program as an input.
  - M(program P,input I)=1 if P(I) halts, else 0.
  - Then let StopsOnSelf(P)=M(P,P). Call it S.
  - Let H(P) = if S(P)=1 do infinite loop else return 1;

  - Now, consider H(H). H(H)=1 iff S(H)=0 iff H runs forever. But H runs forever iff S(H)=1 iff H does not return 1. This is a contradiction!

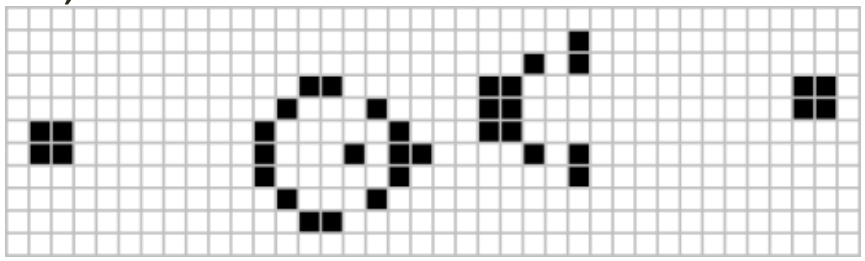
- Conclusion. The assumption M exists is wrong.

EXAMPLES

Conway's Game of Life

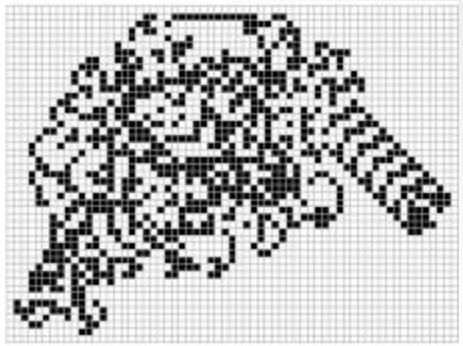


- October 1970 Scientific American
- infinite 2D grid of square *cells*, each *alive* or *dead*.
- 8 *neighbors*
- At each step in time
  - Any live cell
    - with fewer than two live neighbors dies, by under-population.
    - with 2 or 3 live neighbors lives on to the next generation.
    - with more than 3 live neighbors dies, by overcrowding.
  - Any dead cell
    - with exactly 3 live neighbors becomes a live cell, by reproduction.



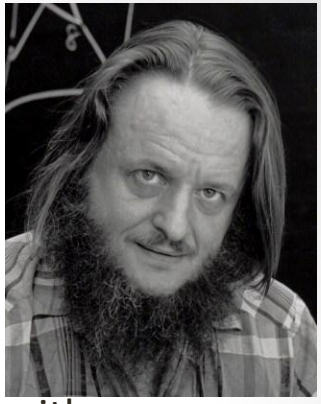
- Life is equivalent to a Universal Turing Machine!

Langton's Ant



- Grid with squares colored white or black.
- "Ant" has direction N,S,E,W.
- Rules:
  - At a white square, turn 90° right, flip the color of the square, move forward one unit
  - At a black square, turn 90° left, flip the color of the square, move forward one unit
- Known to be Turing complete.

Fractran



- Invented by John Conway.
- A FRACTRAN program is an ordered list of positive fractions together with an initial positive integer input n.
- The program is run by updating the integer n as follows:
  - for the first fraction f in the list for which nf is an integer, replace n by nf
  - repeat this rule until no fraction in the list produces an integer when multiplied by n, then halt.
- Conway gave an interesting formula for primes in FRACTRAN:

$$\left(\frac{17}{91}, \frac{78}{85}, \frac{19}{51}, \frac{23}{38}, \frac{29}{33}, \frac{77}{29}, \frac{95}{23}, \frac{77}{19}, \frac{1}{17}, \frac{11}{13}, \frac{13}{11}, \frac{15}{14}, \frac{15}{2}, \frac{55}{1}\right)$$

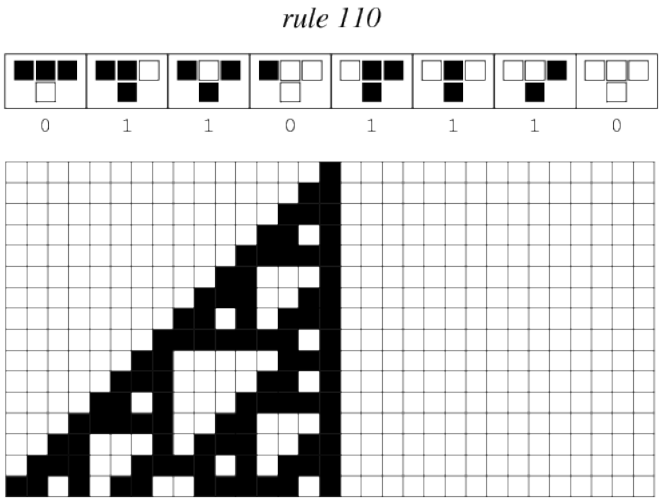
- Starting with *n*=2, this FRACTRAN program generates the following sequence of integers:
- 2, 15, 825, 725, 1925, 2275, 425, 390, 330, 290, 770, ... After 2, this sequence contains the following powers of 2:

$$2^2 = 4,\ 2^3 = 8,\ 2^5 = 32,\ 2^7 = 128,\ 2^{11} = 2048,\ 2^{13} = 8192,\ 2^{17} = 131072,\ 2^{19} = 524288,\ \ldots$$

- **FRACTRAN** is a Turing-complete esoteric programming language invented by the mathematician

Rule 110

- Popularized by Wolfram's NKS
- Shown to Turing Complete



Busy Beaver

- **A Busy Beaver**
  - Turing machine that attains maximum "busyness" among all Turing machines in a certain class
    - measured by the number of steps performed, or the number of nonblank symbols finally on the tape.
- Busy Beaver function Σ: $N \rightarrow N$, $\Sigma(n)$ is the maximum number of 1s finally on the tape among all halting 2-symbol n-state Turing machines when started on a blank tape.
- The infinite sequence **Σ** is the **busy beaver function.**
  - **Σ is non-computable.**
  - Known: $\Sigma(0) = 0$, $\Sigma(1) = 1$, $\Sigma(2) = 4$, $\Sigma(3) = 6$, $\Sigma(4) = 13$.
  - $\Sigma(n)$ not yet been determined for any instance of $n > 4$

Church-Turing Thesis

- **"everything algorithmically computable is computable by a Turing machine."**
- Very powerful, probably not quite true
- Dozens to hundreds of computing “constructs” have been shown to be equivalent.

- Computable functions are exactly the functions that can be calculated using a mechanical calculation device given unlimited amounts of time and storage space.

- Shows equivalence of
  - Turing machines
  - Lambda Calculus
  - Register machines
  - mu-recursive functions

Attempts to surpass

- Allow infinitely many simultaneous changes
- Allow an oracle that can answer a specific problem, such as a Halting Problem Answering oracle
- Allow computation on arbitrary real numbers (instead of the computable reals).
- Allow Closed Timelike Curves from physics
- Allow Malament-Hogarth spacetime or black holes
- Quantum mechanical computation (beyond the current model, which is Turing equivalent).
- Allow arbitrarily small components

- So far, nothing *physically realizable* has given a stronger model

LIMITS TO KNOWLEDGE

Gödel Incompleteness

- Is it possible to find a complete and consistent set of axioms for all mathematics?
- Kurt Gödel, 1931, proved it is impossible. It was a major shock.
- First Incompleteness Theorem:
  - Any system powerful to include the integers (and multiplication) **will** contain a true statement that cannot be proven true with the axioms.
- Second Incompleteness Theorem
  - Such a system cannot demonstrate its own consistency.
- Thus it is impossible to find a complete and consistent set of axioms for all mathematics
- End to Leibniz Dream.

Ax. 1. $P(\varphi) \wedge \Box\, \forall x[\varphi(x) \rightarrow \psi(x)] \rightarrow P(\psi)$
Ax. 2. $P(\neg\varphi) \leftrightarrow \neg P(\varphi)$
Th. 1. $P(\varphi) \rightarrow \Diamond\, \exists x\, [\varphi(x)]$
Df. 1. $G(x) \iff \forall\varphi[P(\varphi) \rightarrow \varphi(x)]$
Ax. 3. $P(G)$
Th. 2. $\Diamond\, \exists x\, G(x)$
Df. 2. $\varphi \operatorname{ess} x \iff \varphi(x) \wedge \forall\psi\{\psi(x) \rightarrow \Box\, \forall x[\varphi(x) \rightarrow \psi(x)]\}$
Ax. 4. $P(\varphi) \rightarrow \Box\, P(\varphi)$
Th. 3. $G(x) \rightarrow G \operatorname{ess} x$
Df. 3. $E(x) \iff \forall\varphi[\varphi \operatorname{ess} x \rightarrow \sqcap\, \exists x\, \varphi(x)]$
Ax. 5. $P(E)$
Th. 4. $\Box\, \exists x\, G(x)$

Gödel Incompleteness II

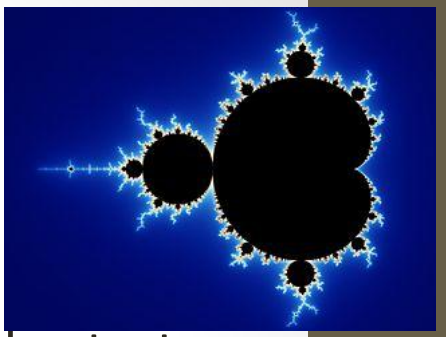
- Proof works by encoding rules of deduction

$=$	$\vee$	$\wedge$	$\neg$	$\rightarrow$	$\forall$	$\exists$	$+$	$\cdot$	$<$	0	1	(	)
1	3	5	7	9	11	13	15	17	19	21	23	25	27

  - Assign numbers to steps needed, build up the equivalent of
    - "This statement is false"
  - Leads to a contradiction.

- Translate mathematical statements into integers, using primes to powers to encode statements, similar to enumerating all programs
- Basically encodes "this statement is false", which can be neither a true nor false statement.
- Says: "G is not provable in theory T".

Kolmogorov Complexity



- the complexity of a string is the length of the string's shortest description in some fixed universal description language.
- If a description of *s*, *d*(*s*), is of minimal length—i.e. it uses the fewest number of characters—it is called a **minimal description** of *s*. Then the length of *d*(*s*)—i.e. the number of characters in the description—is the **Kolmogorov complexity** of *s*, written *K*(*s*). Symbolically, $K(s) = |d(s)|$
- **Theorem**. If K_1 and K_2 are the complexity functions relative to description languages L_1 and L_2, then there is a constant *c* (which depends only on the languages L_1 and L_2) such that
- For all s, $|K1(s) - K2(s)| < c$.
- **Theorem**. There is a constant *c* such that for all s, $K(s) < |s| + c$.
- **Theorem**. K is not computable.

Chaitin Constant

- **Chaitin constant** (**Chaitin omega number**) or **halting probability**
  - real number that represents the probability that a randomly constructed program will halt.
  - Each halting probability is a normal and transcendental real number which is not computable, which means that there is no algorithm that enumerates its digits.

- Chaitin Incompleteness
  - **Theorem**. There exists a constant *L* (which only depends on the particular axiomatic system and the choice of description language) such that there does not exist a string *s* for which the statement $K(s) \geq L$ (as formalized in **S**) can be proven within the axiomatic system **S**.

Chaitin Constant

- Each Chaitin constant Ω has the following properties:
  - algorithmically random: the shortest program to output the first *n* bits of Ω must be of size at least *n*-O(1).
  - normal number: its digits are equidistributed.
  - not a computable number; there is no computable function that enumerates its binary expansion.
  - The set of rational numbers *q* such that $q > \Omega$ is not computably enumerable.
  - It is Turing equivalent to the halting problem and thus at level of the arithmetical hierarchy.

Turing Degree

- Every Turing degree is countably infinite, that is, it contains exactly $\aleph_0$ sets.
- There are $2^{\aleph_0}$ distinct Turing degrees.
- For each degree **a** the strict inequality **a** < **a**' holds.
- For each degree **a**, the set of degrees below **a** is at most countable. The set of degrees greater than **a** has size $2^{\aleph_0}$.
- **Order properties**
- There are **minimal degrees**. A degree **a** is *minimal* if **a** is nonzero and there is no degree between **0** and **a**. Thus the order relation on the degrees is not a dense order.
- For every nonzero degree **a** there is a degree **b** incomparable with **a**.
- There is a set of pairwise incomparable Turing degrees.
- There are pairs of degrees with no greatest lower bound. Thus is not a lattice.
- Every countable partially ordered set can be embedded in the Turing degrees.
- No infinite, strictly increasing sequence of degrees has a least upper bound.

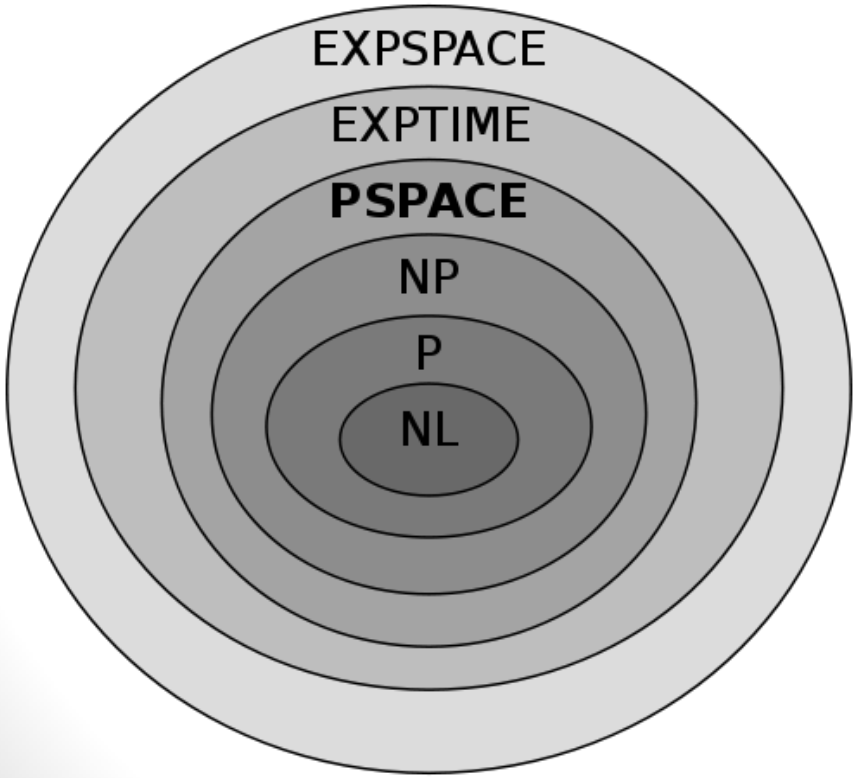
ALGORITHM EFFICIENCY

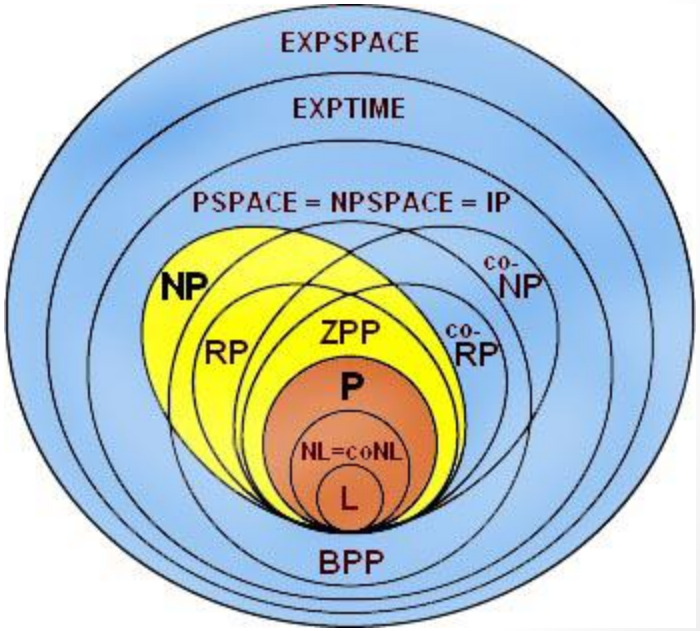
Efficiency of solution

- Addition – takes n steps for n digit numbers
- Before around 1960-1970, the notion of complexity was vague, and many thought most problems would find efficient solutions eventually.
- Some problems require proven amounts of “work”
  - Comparison based sorting requires in the worst case O(n log n) comparisons
  - FFT requires O(n log n) work
  - Binary search requires O(log n) steps

Complexity Class Zoo

- Each is a class of problems and the requirements needed to solve them
- 100's of them in the literature





P and NP

- **P**: all decision problems solvable on a deterministic TM using **p**olynomially many steps in the size of the input.
  - Includes most algorithms in use
- **NP**: non-deterministically polynomial time. All decision problems such that an answer of "yes" can be checked to be "yes" in polynomial time.
  - Includes graph isomorphism, boolean satisfiability, traveling salesman problem
  - Stephen Cook published 1971 twenty-one NP-complete problems, won a Turing award (like a Nobel for CS).
- **P** is contained in **NP**
- Most important problem in computer science : does **P=NP**?
  - worth at least a million $, and possibly billions of $, to resolve

Open Problems

- One of the central questions of complexity theory is whether randomness adds power; that is, is there a problem which can be solved in polynomial time by a probabilistic Turing machine but not a deterministic Turing machine?

- Some scientists are starting to think fundamental computational laws such as P!=NP are laws of physics, like the second law of Thermodynamics.

- Incompleteness as a limit to a universal physical theory of everything?

EXTENSIONS

Probabilistic TM

- At each step, a source of randomness allows choosing between multiple states
- A central questions of complexity theory:
  - Does randomness adds power? (is there a problem which can be solved in polynomial time by a probabilistic TM but not a deterministic TM?)
- Randomness leads to many efficient algorithms.

Quantum Computing

- Allows factoring integers significantly faster than any known classical algorithm
  - Breaks RSA
- Only solves problems a TM can solve
  - Some problems are provably exponentially more efficient in QC
- Does not disprove Church-Turing thesis
- Possible other physical extensions using gravity might allow more powerful computers, but nothing known

Chaos Computing

- Use a chaotic system to differentiate between close initial states, allowing a quantum computer to perform vastly better
- Not physically realizable due to the need for arbitrarily percise measurements

HyperComputation

- Allow operations beyond Turing Machines
  - Not required to be physical
- Some methods were covered earlier like letting infinitely many locations in a TM change at each step.
- Nothing realizable.

Church-Turing Thesis II

- When efficiency is taken in to a account,
  - "*A probabilistic Turing machine can efficiently simulate any realistic model of computation.*“
- Consequently, the **Quantum Complexity-Theoretic Church–Turing thesis** states:
  - ***"A quantum Turing machine can efficiently simulate any realistic model of computation."***

Questions?