So far in this course:

- algorithms, efficiency, polynomial vs. exponential time

Bad news: some problems seem to have only exponential time algorithms (NP-complete problems)

Next in this course:

Worse news: some problems have no algorithm.

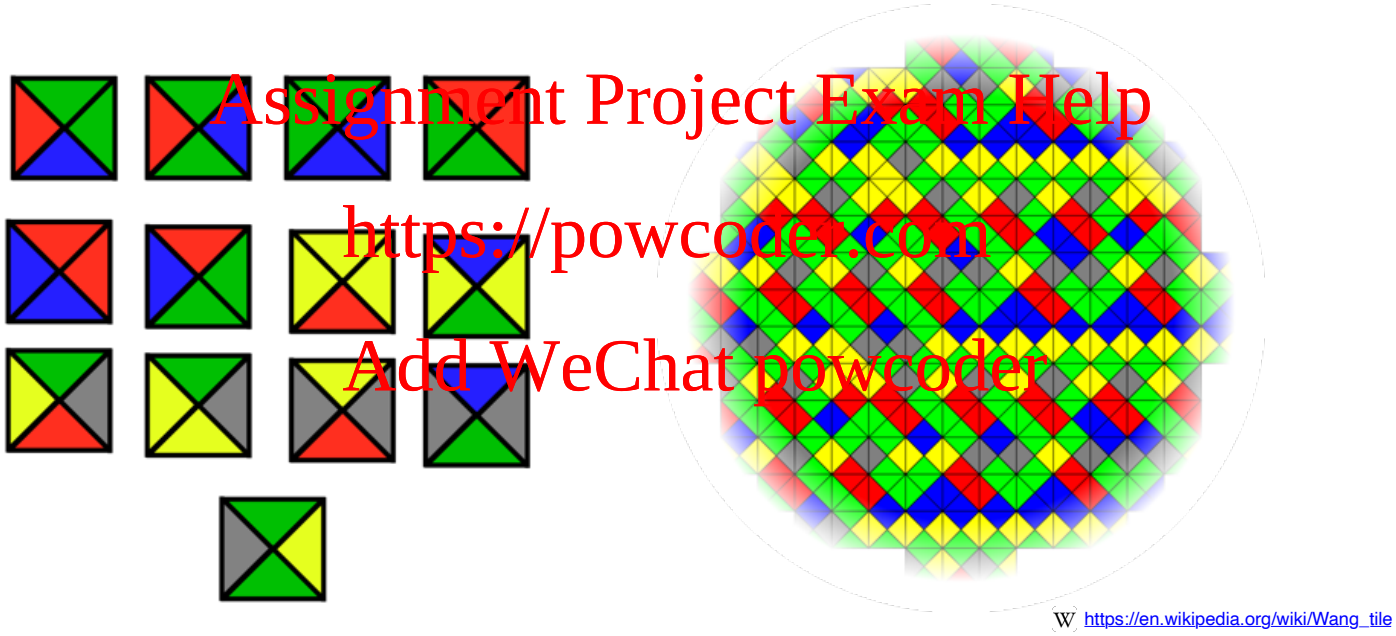
Examples:

1. Program equivalence. Given two programs, do they do the same thing? (i.e., produce the same output for the same input)

   Grad student TAs would love to have this to check if student programs are equivalent to the model solution!

Examples of problems that have no algorithm

2. Tiling. Given a finite set of tiles with coloured edges, can they tile the plane? Colours must match when tiles touch. Tiles cannot be rotated. You can use infinitely many copies of each tile.



https://en.wikipedia.org/wiki/Wang_tile

These tiles tile the plane but only aperiodically.
"Wang" tiles, due to Hao Wang, 1961.

Examples of problems that have no algorithm

3. the Halting Problem. Given a computer program, does it halt?

on all inputs?
some inputs?
specific inputs?
This will be clarified.

e.g.
```
while n ≠ 1 do
   n := n – 2
end
```

This program halts if n is an odd positive number.

e.g.
```
while n ≠ 1 do
   if n is even then n := n / 2
   else n := 3n + 1
end
```


Does this program halt on all natural number inputs? No one knows.
Empirical evidence: the program has halted on every input tried.

12, 6, 3, 10, 5, 16, 8, 4, 2, 1 halt.

"3n + 1 problem" https://en.wikipedia.org/wiki/Collatz_conjecture

We can convert any math statement about existence to a question about a program halting.
This means that questions about a program halting can be very deep and hard.

"Does there exist a number n such that FOO(n) holds?"
turns into the program:

```
n := 1
while not FOO(n) do
   n := n + 1
end
```


e.g. Goldbach's conjecture, 1742. Is every even number the sum of two primes?
i.e., Does there exist a number 2n that is not the sum of two primes?

known to hold for all $n < 4 \times 10^{18}$ but open for general n

When the existence question is open then no one knows whether the corresponding program halts.

Definition. A decision problem is ***undecidable*** if it has no algorithm.

Definition. (more general, not just for decision problems)
A problem is ***unsolvable*** if it has no algorithm.

What is an algorithm?

As discussed earlier in the course, any of the following, all equivalent

- Java / C / Python programs

- pseudo-code

- Random Access Machine (RAM)

- Turing machine

We will study some undecidable problems.

History of undecidability.

Gottlob Frege (1848 - 1925)

Tried to put all of mathematics on a firm foundation — to formulate a set of axioms (e.g., axioms of Peano arithmetic, axioms of set theory) such that every true mathematical statement could be proved from the axioms using basic rules of logic.

His plan was foiled by Russell's paradox.

History of undecidability.

Russell's paradox.

Let S = the set of all sets that do not contain themselves
(this is like the bibliography of all bibliographies that do not list themselves, or the web pages that link to web pages that do not link to themselves)

Then, is S a member of itself?

Either answer (YES or NO) gives a contradiction.

Once a system is powerful enough, you can get self-references that lead to contradictions.

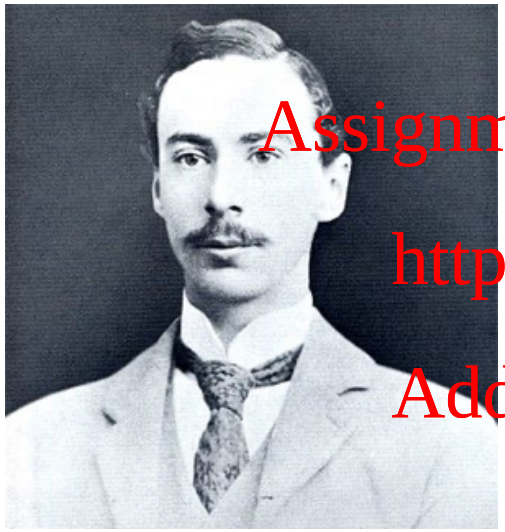
Russell's paradox arises because for Set Theory, you need to define a set. Is a set just a collection of things? — That's too powerful! It leads to Russell's paradox. Moral (in this case) — we must define sets more carefully.

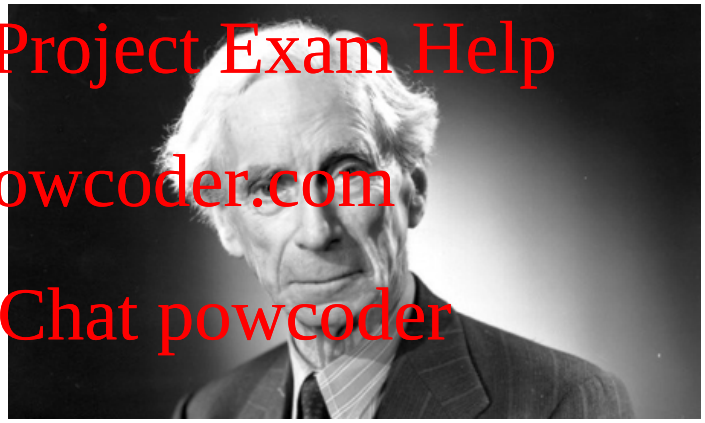
How does the idea of Russell's paradox apply to computing?
It allows us to prove that there are undecidable problems.
If algorithms were powerful enough to decide all problems then we could decide something self-referential, and this gives a contradiction.

History of undecidability.

Bertrand Russell (1872 - 1970)

https://en.wikipedia.org/wiki/Bertrand_Russell





polymath, philosopher, logician, mathematician, historian, writer, social critic, political activist, Nobel laureate

FREGE ON RUSSELL'S PARADOX

Grundgesetze der Arithmetik, Vol. ii, Appendix, pp. 253-65

HARDLY anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished.

This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion. It is a matter of my Axiom (V).[A] I have never disguised from myself its lack of the self-evidence that belongs to the other axioms and that must properly be demanded of a logical law. And so in fact I indicated this weak point in the Preface to Vol. i (p. VII). I should gladly have dispensed with this foundation if I had known of any substitute for it. And even now I do not see how arithmetic can be scientifically established; how numbers can be apprehended as logical objects, and brought under review; unless we are permitted—at least conditionally—to pass from a concept to its extension. May I always speak of the extension of a concept—speak of a class? And if not, how are the exceptional cases recognized? Can we always infer from one concept's coinciding in extension with another concept that any object that falls under the one falls under the other likewise? These are the questions raised by Mr. Russell's communication.

Solatium [*sic*] *miseris socios habuisse malorum*. I too have this comfort, if comfort it is; for everybody who has made use in his proofs of extensions of concepts, classes, sets,* is in the same position as I. What is in question is not just my particular way of establishing arithmetic, but whether arithmetic can possibly be given a logical foundation at all.

But let us come to the point. Mr. Russell has discovered a contradiction which may now be stated.

* Herr R. Dedekind's 'systems' also come under this head.

[A] Vol. i, § 3, § 20. Cf. also Frege's *Function und Begriff* for an explanation of the ideas used; especially pp. 9–10, 18. For any (first-level) function of one argument, there is some object that is its *value-range*; and two such functions are by Axiom (V) equal in value-range if and only if their values always equal for any given argument. Concepts (ibid., pp. 15–16) are functions whose values can only be the True or the False. For the value-ranges of concepts, which are called their *extensions*, the principle runs thus: Two concepts are equal in extension if and only if whatever falls under either falls under the other.

Nobody will wish to assert of the class of men that it is a man. p. 254] We have here a class that does not belong to itself. I say that something belongs to a class when it falls under the concept whose extension the class is. Let us now fix our eye on the concept: *class that does not belong to itself*. The extension of this concept (if we may speak of its extension) is thus the class of classes that do not belong to themselves. For short we will call it the class K. Let us now ask whether this class K belongs to itself. First, let us suppose it does. If anything belongs to a class, it falls under the concept whose extension the class is. Thus if our class belongs to itself, it is a class that does not belong to itself. Our first supposition thus leads to self-contradiction. Secondly, let us suppose our class K does not belong to itself; then it falls under the concept whose extension it itself is, and thus does belong to itself. Here once more we likewise get a contradiction!

What attitude must we adopt towards this? Must we suppose that the law of excluded middle does not hold good for classes? Or must we suppose there are cases where an unexceptionable concept has no class answering to it as its extension? In the first case we should find ourselves obliged to deny that classes are objects in the full sense. For if classes were proper objects, the law of excluded middle would have to hold for them. On the other hand, there is nothing 'unsaturated' or predicative about classes that would clearly debar them as functions, concepts, or relations. What we usually consider as a name of a class, e.g. 'the class of prime numbers,' has rather the nature of a proper name; it cannot occur predicatively, but *can* occur as the grammatical subject of a singular proposition, e.g. 'the class of prime numbers contains infinitely many objects.' If we were going to dispense classes from the law of excluded middle, we might think of regarding them (and, in fact, value-ranges generally) as improper objects. These could then not be allowed as arguments for all first-level functions. But there would also be functions that could have as arguments both proper and improper objects. At least the relation of equality (identity) would be a function of this sort. (An attempt might be made to escape this by assuming a special sort of equality for improper objects. But that is certainly ruled out. Identity is a relation given to us in such a specific form that it is inconceivable that various kinds of it should occur.)

History of undecidability.

In 1928 David Hilbert asked:

- Is mathematics **complete**, i.e., every statement can either be proved or disproved.

- Is mathematics **consistent**, i.e., no valid sequence of proof steps gives a contradiction such as 0 = 1.

- Is mathematics **decidable**, i.e., is there an algorithm to determine the truth of any statement.
  "Entscheidungsproblem" (= decision problem)

David Hilbert (1862 - 1943)

In 1931 Gödel showed:

- every sufficiently powerful mathematical system is either inconsistent or incomplete

- such a system cannot be proved consistent within its own axioms

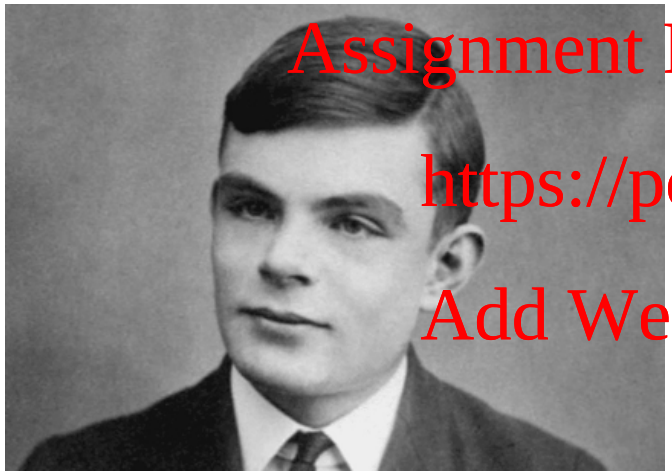
The third question remained open.

Kurt Gödel (1906 - 1978)

History of undecidability.

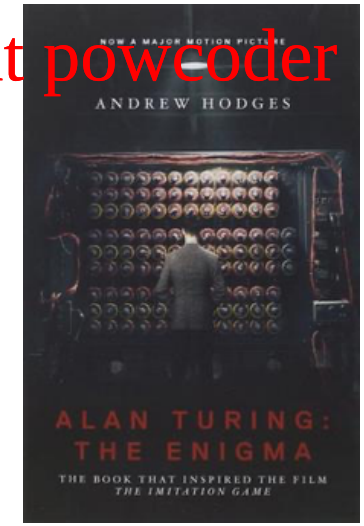
Hilbert's third question — is there an algorithm to determine the truth of any mathematical statement?

was answered by Alan Turing in 1931. NO, there is no such algorithm.



Alan Turing (1912 - 1954)

https://en.wikipedia.org/wiki/Alan_Turing



Alan Turing: The Enigma

The Imitation Game

Turing's paper

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

Summary of Lecture 24, Part 1

history and concept of undecidability

What you should know

- definition of undecidable

Next:

proving that problems are undecidable

Undecidability proofs.

different

As with NP-completeness, the first undecidability proof is ~~hard~~, but the following undecidability proofs use reductions.
The first problem we will show is undecidable:

The Halting Problem. Given an algorithm/program A and input w, does A halt on w?

Theorem. The Halting Problem is undecidable.
Proof.
Main idea: self-reference to get a contradiction as in Russell's paradox.

Suppose, for a contradiction, that there is a program H for the halting problem.

Input for H: program A and input w
Output: YES/NO, does A halt on w

Can we have a program as input?
Sure, compilers do it all the time.

Using H, we can make a new program H'

```
Input for H': a program B
      begin
         call H(B,B)
         if it returns NO then HALT
         else loop forever
      end
```

H' is like Russell's set S. Russell's question (Does S contain S?) becomes . . .

repeated from previous page

Suppose, for a contradiction, that there is a program H for the halting problem.

Input for H: program A and input w

Output: YES/NO, does A halt on w

Can we have a program as input?
Sure, compilers do it all the time.

Using H, we can make a new program H'

```
Input for H': a program B
    begin
      call H(B,B)
      if it returns NO then HALT
      else loop forever
    end
```


H' is like Russell's set S. Russell's question (Does S contain S?) becomes . . .

Does H' halt on input H'?

H' halts on input H'
iff H(H', H') returns YES by definition of H.
iff (looking at code of H')
H' on input H' loops forever
Contradiction.
So our assumption that H exists is wrong.
∴ there is no algorithm to decide the Halting Problem.

Undecidability proofs.

To show that other problems are undecidability — use reductions.

Recall

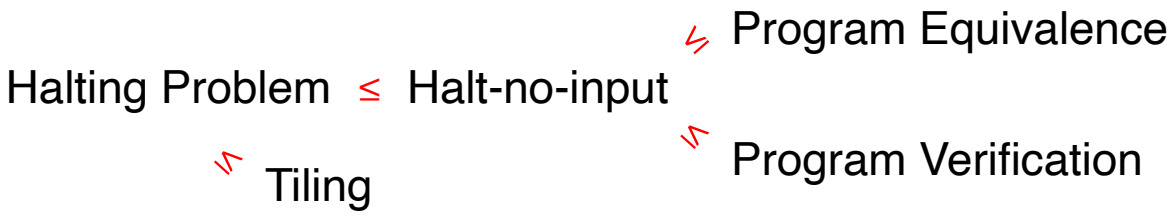
Definition. Problem X ***reduces to*** problem Y, written X ≤ Y, if an algorithm for Y can be used to make an algorithm for X. Think of this as "X is easier than Y".

i.e., if Y has an algorithm, then so does X

Contrapositive:

if X is undecidable, then Y is undecidable

Road map of undecidability results:



More undecidabe problems.

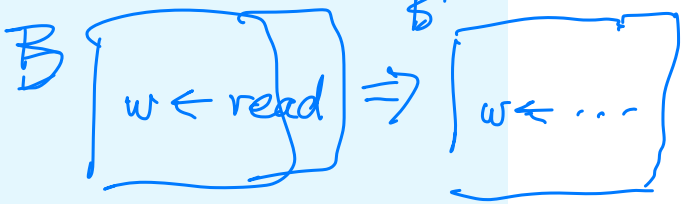
Halt-no-input. Given a program A (no input), does A halt?
Theorem. Halt-no-input is undecidable.
Proof. Show that: Halting problem ≤ Halt-no-input

Assume we have an algorithm P that decides Halt-no-input.
Input for P: program A
Output: YES/NO does A halt

Build an algorithm to decide the Halting problem
Input: program B, input w
Output: YES/NO does B halt on input w?

begin
- modify the code of B to make a program B' that has w hard coded inside it and runs B on w.
- call P(B')
- output the YES/NO answer.

end



Observe: B' halts (no input) iff B halts on input w
Thus the algorithm is correct.
(No need to do runtime analysis)

More undecidabe problems.

Program Verification. Given a program and specification of the inputs and corresponding outputs, does the program meet the specifications?

Theorem. Program Verification is undecidable.
So automatic program checking is impossible.

Note that we need a *finite* specification on inputs/outputs.

Proof. Show that: Halt-no-input ≤ Program Verification.

Assume we have an algorithm V that decides Program Verification.
Input for V: program + input/output specs
Output: YES/NO does the program meet the specs

Build an algorithm to decide Halt-no-input
Input: program A
Output: YES/NO does A halt?

modify program A as follows [don't run it, just edit it]

program A' { read input (and ignore it)
code of A
output 1

Call V(A', specs: for every input, output is 1)
output YES/NO answer

Correctness: A' outputs 1 on all inputs iff A halts.

More undecidabe problems.

Program Equivalence. Given two programs, do they do the same thing (i.e., produce same output for same input?

Theorem. Program Equivalence is undecidable.
Proof. Show that: Halt-no-input ≤ Program Equivalence.

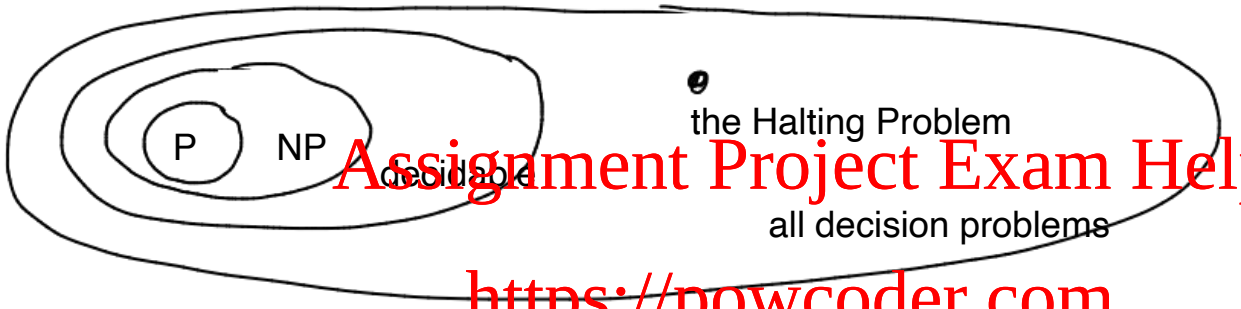
The reduction is very similar to the above.
Construct A' as above.
Construct program B: read input (and ignore it)
output 1

Then ask if A' and B are equivalent.
They are equivalent iff A halts.

Ex. Fill in the details.

There are many more interesting undecidable problems.

Finally, to connect this topic with the earlier part of the course:
Are there interesting classes between NP and decidable?



$P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq$ decidable $\subseteq$ all decision problems

polynomial space

run time is $O(2^{poly(n)})$

https://en.wikipedia.org/wiki/Time_complexity#Exponential_time

known: $P \neq EXP$, but all else is open.

OPEN. Is P = PSPACE?

Where to learn more about undecidability and complexity classes: CS 360 and CS 365.

Where to learn more about algorithms: CS 466, Combinatorial Optimization.

Summary of Lecture 24

undecidability
- definition
- examples of undecidable problems
- proofs of undecidability

What you should know

- definition of undecidable

- how to prove a problem is undecidable (use Turing reduction $\leq$)