The Halting Problem

There is a specific problem that is algorithmically unsolvable!

- One of the most philosophically important theorems in the theory of computation
- In fact, ordinary/practical problems may be unsolvable
- Software verification: Given a computer program and a precise specification of what the program is supposed to do (e.g., sort a list of numbers). Come up with an algorithm to prove the program works as required
    - This cannot be done!
    - But wait, can't we prove an addition, multiplication, sorting algorithm works?
    - Note: The proof is not only to prove it works for a specific task, like sorting numbers but that its behavior always follows the specification!

The first undecidable problem

Does a TM accept a given input string?

- We have shown that a CFL is decidable and a CFG can be simulated by a TM.
- This does not yield a contradiction. TMs are more expressive than CFGs.

Why “Halting” problem?

- A_{TM} = {(M,w)|M is a TM and M accepts w}
- A_{TM} is undecidable
  - It can only be undecidable due to a loop of M on w.
  - If we could determine if it will loop forever, then could reject. Hence A_{TM} is often called the halting problem.
    - As it is impossible to determine if a TM will always halt on every possible input
  - Note that this is Turing recognizable! We can simulate M on input w
    - If M accepts w then accept (M,w)
    - If M rejects w then reject (M,w)

Comparison of infinite sets

In 1873 mathematician Cantor was concerned with the problem of measuring the sizes of infinite sets

- How can we tell if one infinite set is bigger than another or if they are the same size?
  - We cannot use the counting method that we would use for finite sets. Example: how many even integers are there?
- Cantor observed that two finite sets have the same size if each element in one set can be paired with the element in the other

Function Property Definitions

Given a set A and B and a function *f* from A to B

- *f* is one-to-one if it never maps two elements in A to the same element in B
  - The function *add-two* is one-to-one whereas *absolute-value* is not
- *f* is onto if every item in B is reached from some value in a (i.e., f(a) = b for every b $\in$ B).
  - For example, if A and B are the set of integers, then *add-two* is onto but if A and B are the positive integers, then it is not onto since b = 1 is never hit.
- A function that is one-to-one and onto has a (one-to-one) correspondence
  - This allows all items in A and B to be paired

An Example of Pairing Set Items

- Let N be the set of natural numbers {1, 2, 3, …} and let E be the set of even natural numbers {2, 4, 6, …}.
- Using Cantor's definition of size we have that N and E have the same size.
  - The correspondence f from N to E is f(n) = 2n.
- This is somehow counter intuitive since E is a proper subset of N!!
- Focus on on the definition: since f(n) is a 1:1 correspondence, so we say they are the same size.
- Definition: A set is *countable* if either it is finite or it has the same size as N, the set of natural numbers (*infinitely countable*)

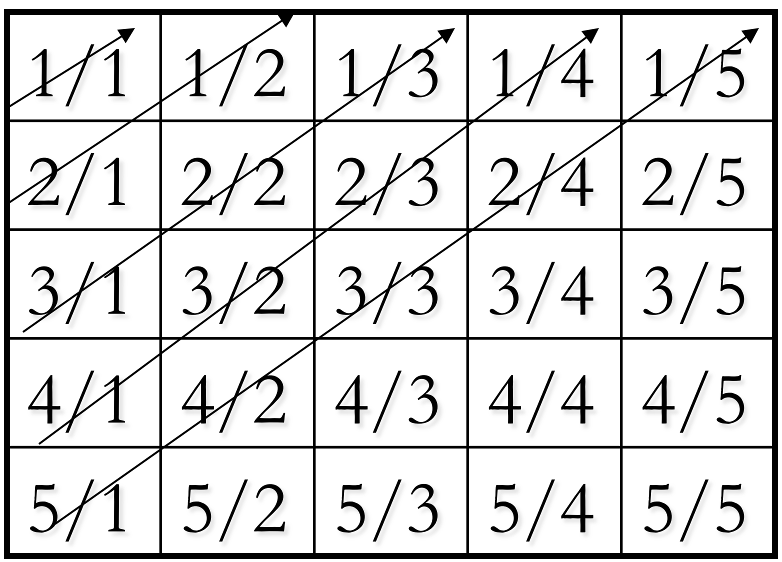
Example: Rational Numbers

- Q = {m/n: m,n ∈ N}, the set of positive Rational Numbers
- Q seems much larger than N, but according to our definition, they are the same size.
  - Here is the 1:1 correspondence between Q and N
  - We need to list all of the elements of Q and then label the first with 1, the second with 2, etc.
    - We need to make sure each element in Q is listed only once

Correspondence between N and Q

To build our correspondence, we build an infinite matrix containing all the positive rational numbers

– Writing the list by going row-to-row or column by column is a bad idea!
  - Since 1st row is infinite, would never get to the second row
– We use diagonals, not adding the values that are equivalent
  - So the order is 1/1, 2/1, ½, 3/1, 1/3, …
– This yields a correspondence between Q and N
  - That is, N=1 corresponds to 1/1, N=2 corresponds to 2/1, N=3 corresponds to ½ etc.



R is Uncountable

- A real number is one that has a decimal representation and R is set of Real Numbers
  - Includes those that cannot be represented with a finite number of digits (e.g., $\pi, \sqrt{2}, 3.\bar{3}$)
- Will show that there can be no *pairing* -no possible one-to-one correspondence- of elements between R and N
  - Proof by contradiction: Given any possible paring we will find some value x that not in the pairing

R is Uncountable

- Assume that one complete mapping exits
- We now describe a recipe to obtain a value x between 0 and 1 which is not in the infinite list
  - To ensure that x ≠ f(1), pick a digit not equal to the first digit after the decimal point. Any value not equal to 1 will work. Pick 4 so we have .4
  - To x ≠ f(2), pick a digit not equal to the second digit. Any value not equal to 5 will work. Pick 6. We have .46
  - Continue, choosing values along the "diagonal" of digits (i.e., if we took the f(n) column and put one digit in each column of a new table).
- The selected value x is guaranteed to not already be in the list since it differs in at least one position with every other number in the list.

n	f(n)
1	3.<u>1</u>4159…
2	55.5<u>5</u>55…
3	0.12<u>3</u>45…
4	0.500<u>0</u>00
.	.

Implications

R being uncountable has an important application in the theory of computation

- There are countably many Turing Machines
- There are uncountably many languages
- Each TM recognizes one single language

→ some languages are not recognized by any Turing machine.

  - Corollary: some languages are not Turing-recognizable

Some Languages are Not Turing-recognizable

Proof:

- The set Σ^* is countable: there are only a finite number of strings of each length, we may form a list of Σ^* by writing down all strings of length 0, length 1, length 2, etc.
- The set of all Turing Machines M is countable since each TM M has an encoding into a string <M>
  - The set of valid TM's is a subset of the set of possible strings.
  - As the latter is countable, so is the former.
- The set of all languages L over Σ is uncountable
  - the set of all infinite binary sequences B is uncountable (each sequence is infinitely long)
    - The same diagonalization proof we used to prove R is uncountable
  - L is uncountable because it has a correspondence with B
    - Assume $\Sigma^* = \{s_1, s_2, s_3 \ldots\}$. We can encode any language as a characteristic binary sequence, where the bit indicates whether the corresponding s_i is a member of the language. Thus, there is a 1:1 mapping.
  - Since B is uncountable and L and B are of equal size, L is uncountable
- Since the set of TMs is countable and the set of languages is not, we cannot put the set of languages into a correspondence with the set of Turing Machines. Thus there exists some languages without a corresponding Turing machine

Halting Problem is Undecidable

Prove that halting problem is undecidable

- Let A_{TM} = {<M,w>| M is a TM and accepts w}
- Proof Technique:
  - Assume A_{TM} is decidable and obtain a contradiction
  - A diagonalization argument

Proof: Halting Problem is Undecidable

- Assume A_{TM} is decidable
- Let H be a decider for A_{TM}
  - On input <M,w>, where M is a TM and w is a string, H halts and accepts if M accepts w; otherwise it rejects
- Construct a TM D using H as a subroutine
  - D calls H(M,<M>) to determine what M does when the input string is its own description <M>.
  - D then outputs the opposite of H's answer
  - D(<M>) accepts if M does not accept <M> and rejects if M accepts <M>
- Assume we run D on its own description D(<D>)
  - D invokes H(D,<D>) which accepts if D accepts <D>; otherwise it rejects
  - D(<D>) = accept if D does not accept <D> and reject if D accepts <D>
  - A contradiction so H cannot be a decider for A_{TM}

Constructing D by diagonalization

	<M1>	<M2>	<M3>	<M4>	…	<D>
M1	Accept	Reject	Accept	Reject	…	Accept
M2	Accept	Accept	Accept	Accept	…	Accept
M3	Reject	Reject	Reject	Reject	…	Reject
M4	Accept	Accept	Reject	Reject	…	Accept
.						
D	Reject	Reject	Accept	Accept	…	Contradiction
.						

Software checking

- You write a program, halts(P, X) that takes as input any program, P, and the input to that program, X
  - Your program halts(P, X) analyzes P and returns "yes" if P will halt on X and "no" if P will not halt on X
- You now write a procedure lul(X) with as single instruction:

  lul(X) {a: if halts(X,X) then go to a else halt}

  This program halts if P does not halt on X; otherwise it does halt
- Does lul(lul) halt?
  - It halts if and only if halts(lul,lul) returns no
    - It halts if and only if it does not halt. Contradiction.
- Thus we have proven that you cannot write a program to determine if an arbitrary program will halt or loop

What does this mean?

- The halting problem asks whether we can tell if some TM M will accept an input string
- We are asking if the language below is decidable
  - $A_{TM} = \{(M,w) | M \text{ is a TM and } M \text{ accepts } w\}$
- It is not decidable
  - M is a input variable too!
  - Some algorithms are decidable, like sorting algorithms
- It is Turing-recognizable
  - Simulate the TM on w and if it accepts/rejects, then accept/reject.

Co-Turing Recognizable

- A language is co-Turing recognizable if it is the complement of a Turing-recognizable language
- Theorem: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable
  - If a language L is Turing-recognizable then there exists a TM1 which accepts its strings in finite time
    - If the TM1 does not accept, it may reject or loop (it which case it may be not decidable).
  - If L is co-Turing-recognizable then its complement $\overline{L}$ is Turing-recognizable
    - Hence there exist TM2 which accepts strings from $\overline{L}$ in finite time
    - If a string is accepted by TM2 then we have that it is not in L
  - L is decidable

Complement of A_{TM} is not Turing-recognizable

- If a language is undecidable, then either the language or its complement is not Turing-recognizable!
- $\overline{A_{TM}}$ is not Turing-recognizable

  Proof:
  - We know that A_{TM} is Turing-recognizable but not decidable
  - If $\overline{A_{TM}}$ were also Turing-recognizable, then A_{TM} would be decidable, which it is not
  - Thus $\overline{A_{TM}}$ is not Turing-recognizable