Turing and Gödel

Steve Ziskind

Can a Machine Think

- Computers are everywhere doing everything.There seems to be no limits to what they can do.

Can a Machine Think

- Computers are everywhere doing everything.There seems to be no limits to what they can do.
- In 1950 Alan Turing (The Father of Computer Science) wrote a paper titled: "Can a Machine Think?". Can they become conscious like people?

Can a Machine Think

- Computers are everywhere doing everything.There seems to be no limits to what they can do.
- In 1950 Alan Turing (The Father of Computer Science) wrote a paper titled: "Can a Machine Think?". Can they become conscious like people?
- Some people might deny computers this ability because they "can't enjoy strawberries and cream", can't fall in love, or cannot write a good poem. However: some animals appear to fall in love, and I cannot write a good poem.

Can a Machine Think

- Computers are everywhere doing everything.There seems to be no limits to what they can do.
- In 1950 Alan Turing (The Father of Computer Science) wrote a paper titled: "Can a Machine Think?". Can they become conscious like people?
- Some people might deny computers this ability because they "can't enjoy strawberries and cream", can't fall in love, or cannot write a good poem. However: some animals appear to fall in love, and I cannot write a good poem.
- Because the question of thinking is ill defined and subjective, Turing proposed The Imitation Game, now called The Turing Test.

Can a Machine Think

- Computers are everywhere doing everything.There seems to be no limits to what they can do.
- In 1950 Alan Turing (The Father of Computer Science) wrote a paper titled: "Can a Machine Think?". Can they become conscious like people?
- Some people might deny computers this ability because they "can't enjoy strawberries and cream", can't fall in love, or cannot write a good poem. However: some animals appear to fall in love, and I cannot write a good poem.
- Because the question of thinking is ill defined and subjective, Turing proposed The Imitation Game, now called The Turing Test.
- He said that the best way to test the question of thinking was to see if a machine (i.e. computer) behaved like a person. A machine is thinking if it can provide answers in a discussion that are indistinguishable from those of a person.

Can a Machine Think

- Today no computer can successfully pass the Turing Test against a sophisticated opponent, though naive players are often fooled into thinking that the machine on the other end of the line is a person.

Can a Machine Think

- Today no computer can successfully pass the Turing Test against a sophisticated opponent, though naive players are often fooled into thinking that the machine on the other end of the line is a person.
- Perhaps the real issue is not so much that machines can imitate people, but that some people exhibit mechanical behavior.

Can a Machine Think

- Today no computer can successfully pass the Turing Test against a sophisticated opponent, though naive players are often fooled into thinking that the machine on the other end of the line is a person.
- Perhaps the real issue is not so much that machines can imitate people, but that some people exhibit mechanical behavior.
- Maybe we are asking the wrong question. We expect computers to compute, not behave like people. We should ask what computers cannot compute.

Can a Machine Think

- Today no computer can successfully pass the Turing Test against a sophisticated opponent, though naive players are often fooled into thinking that the machine on the other end of the line is a person.
- Perhaps the real issue is not so much that machines can imitate people, but that some people exhibit mechanical behavior.
- Maybe we are asking the wrong question. We expect computers to compute, not behave like people. We should ask what computers cannot compute.
- There are, of course, many things that a computer cannot compute, and for simple and obvious reasons.

Impossible Computations

- A computer cannot provide a list of all the prime numbers less that $n = 10^{10^{10^6}}$.

Impossible Computations

- A computer cannot provide a list of all the prime numbers less that $n = 10^{10^{10^6}}$.
- Actually this task is easily programmed, but the value of n is so large that the universe will probably come to an end before the calculation finishes. This isn't the type of impossibility that should bother us.

Impossible Computations

- A computer cannot provide a list of all the prime numbers less that $n = 10^{10^{10^6}}$.
- Actually this task is easily programmed, but the value of n is so large that the universe will probably come to an end before the calculation finishes. This isn't the type of impossibility that should bother us.
- Find a pair of positive integers, (m, n) that satisfy the equation $m^2 = 2n^2$. This can't be done because no such pair of integers exists. The ratio m/n would equal $\sqrt{2}$, which is known to be irrational. We're asking the computer to find something that doesn't exist. This isn't a limit on the computer.

Impossible Computations

- A computer cannot provide a list of all the prime numbers less that $n = 10^{10^{10^6}}$.
- Actually this task is easily programmed, but the value of n is so large that the universe will probably come to an end before the calculation finishes. This isn't the type of impossibility that should bother us.
- Find a pair of positive integers, (m, n) that satisfy the equation $m^2 = 2n^2$. This can't be done because no such pair of integers exists. The ratio m/n would equal $\sqrt{2}$, which is known to be irrational. We're asking the computer to find something that doesn't exist. This isn't a limit on the computer.
- We want to find an impossible calculation that seems like it should be possible.

The Halting Problem

- In my student days computers were rare, large, and expensive. Many people wanted access to them, and it wasn't always easy to get.

The Halting Problem

- In my student days computers were rare, large, and expensive. Many people wanted access to them, and it wasn't always easy to get.
- When I took a programming class, each student was allocated 10 CPU seconds for homework.This was plenty if you didn't make a tragic blunder. Without intending to do it, people occasionally wrote a program that went into an infinite loop: it never halted. If you did this your allocated time would expire, you were booted off the machine, and you flunked. (or had to beg for forgiveness or drop the course)

The Halting Problem

- In my student days computers were rare, large, and expensive. Many people wanted access to them, and it wasn't always easy to get.
- When I took a programming class, each student was allocated 10 CPU seconds for homework.This was plenty if you didn't make a tragic blunder. Without intending to do it, people occasionally wrote a program that went into an infinite loop: it never halted. If you did this your allocated time would expire, you were booted off the machine, and you flunked. (or had to beg for forgiveness or drop the course)
- Wouldn't it be great if there were a program that could tell you ahead of time if this would happen? You could give it the inputs of: 1) your program, and 2) the input to your program. It would correctly determine whether your program, when run with your input, would halt, (and would do so without going into an infinite loop itself).

The Halting Problem

- With an argument both very simple and very clever, Turing was able to prove that no such halting test program, HT, could exist. The key idea is to feed a program to itself as the input, with a twist.

The Halting Problem

- With an argument both very simple and very clever, Turing was able to prove that no such halting test program, HT, could exist. The key idea is to feed a program to itself as the input, with a twist.
- Such a supposed HT would take 2 arguments: 1) the program you were worried about, and 2) the input you planned to give to your program. It would look like HT(P,I). If P(I) halted, then HT(P,I) = True. If P(I) ran forever, then HT(P,I)=False.

The Halting Problem

- Turing now used HT to contradict itself. Let a new program C be defined like this: C takes a single program P as input, and if HT(P,P)=True then C(P) intentionally goes into an infinite loop and never halts. On the other hand, if HT(P,P)=False then C(P) immediately halts.

The Halting Problem

- Turing now used HT to contradict itself. Let a new program C be defined like this: C takes a single program P as input, and if HT(P,P)=True then C(P) intentionally goes into an infinite loop and never halts. On the other hand, if HT(P,P)=False then C(P) immediately halts.
- Now we ask: what is HT(C,C)?

The Halting Problem

- Turing now used HT to contradict itself. Let a new program C be defined like this: C takes a single program P as input, and if HT(P,P)=True then C(P) intentionally goes into an infinite loop and never halts. On the other hand, if HT(P,P)=False then C(P) immediately halts.
- Now we ask: what is HT(C,C)?
- If HT(C,C)=True, then C(C) loops.

The Halting Problem

- Turing now used HT to contradict itself. Let a new program C be defined like this: C takes a single program P as input, and if HT(P,P)=True then C(P) intentionally goes into an infinite loop and never halts. On the other hand, if HT(P,P)=False then C(P) immediately halts.
- Now we ask: what is HT(C,C)?
- If HT(C,C)=True, then C(C) loops.
- If HT(C,C)=False, then C(C) halts.

The Halting Problem

- Turing now used HT to contradict itself. Let a new program C be defined like this: C takes a single program P as input, and if HT(P,P)=True then C(P) intentionally goes into an infinite loop and never halts. On the other hand, if HT(P,P)=False then C(P) immediately halts.
- Now we ask: what is HT(C,C)?
- If HT(C,C)=True, then C(C) loops.
- If HT(C,C)=False, then C(C) halts.
- Both of these possible answers are reversed. HT cannot exist.

The Halting Problem

- Strange as the Undecideability of the Halting Problem is, here is something even stranger.

The Halting Problem

- Strange as the Undecideability of the Halting Problem is, here is something even stranger.
- Turing published his result in 1937. Computers were invented during and after WWII, several years after 1937. (by Turing, among others)

The Halting Problem

- Strange as the Undecideability of the Halting Problem is, here is something even stranger.
- Turing published his result in 1937. Computers were invented during and after WWII, several years after 1937. (by Turing, among others)
- **WTF?**

The Halting Problem

- Strange as the Undecideability of the Halting Problem is, here is something even stranger.
- Turing published his result in 1937. Computers were invented during and after WWII, several years after 1937. (by Turing, among others)
- **WTF?**
- As it happens, there is Logic behind it all.

What a Halting Test could do

- An old problem in number theory is the Goldbach Conjecture ($\sim$ 1740): Every even number > 2 is the sum of 2 primes. It is unsolved.

What a Halting Test could do

- An old problem in number theory is the Goldbach Conjecture ($\sim$ 1740): Every even number > 2 is the sum of 2 primes. It is unsolved.
- It is easy to search for a counter example. A simple program can test even number after even number. If the conjecture is false then a counter example will eventually be found, and the program will halt. If the conjecture is true it will search forever.

What a Halting Test could do

- An old problem in number theory is the Goldbach Conjecture ($\sim$ 1740): Every even number > 2 is the sum of 2 primes. It is unsolved.
- It is easy to search for a counter example. A simple program can test even number after even number. If the conjecture is false then a counter example will eventually be found, and the program will halt. If the conjecture is true it will search forever.
- A Halting Test could examine the program and decide the conjecture, without ever running the program. Problems with a Goldbach flavor (disprovable with a counterexample) could all be solved.

What a Halting Test could do

- An old problem in number theory is the Goldbach Conjecture ($\sim$ 1740): Every even number > 2 is the sum of 2 primes. It is unsolved.
- It is easy to search for a counter example. A simple program can test even number after even number. If the conjecture is false then a counter example will eventually be found, and the program will halt. If the conjecture is true it will search forever.
- A Halting Test could examine the program and decide the conjecture, without ever running the program. Problems with a Goldbach flavor (disprovable with a counterexample) could all be solved.
- Of course, some problems are not of this type. The Twin Prime Conjecture asserts that there are infinitely many primes separated by 2. If false it cannot be shown with a counterexample, so a Halting Test would not help for this.

Geometry

- In 300 BC Euclid published The Elements, easily the most successful technical writing in history.

Geometry

- In 300 BC Euclid published The Elements, easily the most successful technical writing in history.
- The Elements was notable for beginning with postulates and rules of logic, and from them deducing all the theorems of geometry.

Geometry

- In 300 BC Euclid published The Elements, easily the most successful technical writing in history.
- The Elements was notable for beginning with postulates and rules of logic, and from them deducing all the theorems of geometry.
- One of the postulates, regarding parallel lines, seemed different in character from the others, and people tried to deduce it from the others. For millennia, without success.

Geometry

- In 300 BC Euclid published The Elements, easily the most successful technical writing in history.
- The Elements was notable for beginning with postulates and rules of logic, and from them deducing all the theorems of geometry.
- One of the postulates, regarding parallel lines, seemed different in character from the others, and people tried to deduce it from the others. For millennia, without success.
- In the 1820's people realized that it could be replaced with other versions, equally consistent with the other postulates.
  Non-Euclidean geometry was born.

Set Theory

- In the late 1800's people attempted to place Set Theory on a firm foundation after learning of strange problems that arose from a loose use of the term "set".

Set Theory

- In the late 1800's people attempted to place Set Theory on a firm foundation after learning of strange problems that arose from a loose use of the term "set".
- For example, the set of thinkable thoughts is itself a thinkable thought (I just thought of it) and is a member of itself.

Set Theory

- In the late 1800's people attempted to place Set Theory on a firm foundation after learning of strange problems that arose from a loose use of the term "set".
- For example, the set of thinkable thoughts is itself a thinkable thought (I just thought of it) and is a member of itself.
- Self referential objects (sets inside of themselves) cause weird paradoxes.

Set Theory

- In the late 1800's people attempted to place Set Theory on a firm foundation after learning of strange problems that arose from a loose use of the term "set".
- For example, the set of thinkable thoughts is itself a thinkable thought (I just thought of it) and is a member of itself.
- Self referential objects (sets inside of themselves) cause weird paradoxes.
- The only way out of this tangle is to restrict the definition of set. The postulates/axioms are the Zermelo/Frankel Axioms (ZF).

Number Theory

- Around 1900 and for several decades afterwards, several attempts were made to put Number Theory (the study of integers) on a formal basis, similar to what Euclid did for geometry.

Number Theory

- Around 1900 and for several decades afterwards, several attempts were made to put Number Theory (the study of integers) on a formal basis, similar to what Euclid did for geometry.
- Starting from postulates (Peano Axioms) it would be possible to deduce all the consequences (aka theorems).

Number Theory

- Around 1900 and for several decades afterwards, several attempts were made to put Number Theory (the study of integers) on a formal basis, similar to what Euclid did for geometry.
- Starting from postulates (Peano Axioms) it would be possible to deduce all the consequences (aka theorems).
- We want 1) all the theorems to be true, and 2) every statement to be either provable or disprovable. Also, the system should be consistent (i.e. nothing can be both proved and disproved).

Gödel

- In 1931 Kurt Gödel stunned the world of mathematics and logic by proving that no axiomatization of number theory could ever meet the desired goals.

Gödel

- In 1931 Kurt Gödel stunned the world of mathematics and logic by proving that no axiomatization of number theory could ever meet the desired goals.
- More precisely, he showed that in any formal system that captured the arithmetic of integers (including addition and multiplication), there are statements that cannot be proved from the axioms, nor can their negations be so proved.

Gödel

- In 1931 Kurt Gödel stunned the world of mathematics and logic by proving that no axiomatization of number theory could ever meet the desired goals.
- More precisely, he showed that in any formal system that captured the arithmetic of integers (including addition and multiplication), there are statements that cannot be proved from the axioms, nor can their negations be so proved.
- Arithmetic has propositions that are undecidable.

Gödel

- In 1931 Kurt Gödel stunned the world of mathematics and logic by proving that no axiomatization of number theory could ever meet the desired goals.
- More precisely, he showed that in any formal system that captured the arithmetic of integers (including addition and multiplication), there are statements that cannot be proved from the axioms, nor can their negations be so proved.
- Arithmetic has propositions that are undecidable.
- Gödel's proof is quite complex.

Turing and Gödel

- It is claimed in several internet articles that a simple proof of the Undecideabilty of Peano Arithmetic can be based on the Undecideability of the Halting Problem.

Turing and Gödel

- It is claimed in several internet articles that a simple proof of the Undecideabilty of Peano Arithmetic can be based on the Undecideability of the Halting Problem.
- Probably there are good and convincing explanations, buried in carefully written books somewhere. None of those on the internet make any sense.

Turing and Gödel

- It is claimed in several internet articles that a simple proof of the Undecideabilty of Peano Arithmetic can be based on the Undecideability of the Halting Problem.
- Probably there are good and convincing explanations, buried in carefully written books somewhere. None of those on the internet make any sense.
- Let me know if you find a convincing explanation.

Richard's Paradox

- Godel's proof uses a strategy mirroring a paradox invented by Jules Richard in 1905.

Richard's Paradox

- Godel's proof uses a strategy mirroring a paradox invented by Jules Richard in 1905.
- Any property of numbers can be described in English. Because any description is of finite length, the properties can be listed and assigned numbers. For example: 1 - the number is odd, 2 - the number is a perfect square, etc.

Richard's Paradox

- Godel's proof uses a strategy mirroring a paradox invented by Jules Richard in 1905.
- Any property of numbers can be described in English. Because any description is of finite length, the properties can be listed and assigned numbers. For example: 1 - the number is odd, 2 - the number is a perfect square, etc.
- Some numbers satisfy their own properties, some don't. Call those that don't Richardian.

Richard's Paradox

- Godel's proof uses a strategy mirroring a paradox invented by Jules Richard in 1905.
- Any property of numbers can be described in English. Because any description is of finite length, the properties can be listed and assigned numbers. For example: 1 - the number is odd, 2 - the number is a perfect square, etc.
- Some numbers satisfy their own properties, some don't. Call those that don't Richardian.
- One such property is that the number is Richardian. So this property has a number, n. That is, the list looks like:
  1 - number is odd
  2 - number is a square
  ...
  n - number is Richardian

Richard's Paradox

- Godel's proof uses a strategy mirroring a paradox invented by Jules Richard in 1905.
- Any property of numbers can be described in English. Because any description is of finite length, the properties can be listed and assigned numbers. For example: 1 - the number is odd, 2 - the number is a perfect square, etc.
- Some numbers satisfy their own properties, some don't. Call those that don't Richardian.
- One such property is that the number is Richardian. So this property has a number, n. That is, the list looks like:
  1 - number is odd
  2 - number is a square
  ...
  n - number is Richardian
- Now ask if n is Richardian. It immediately is seen that: it is exactly when it is not.