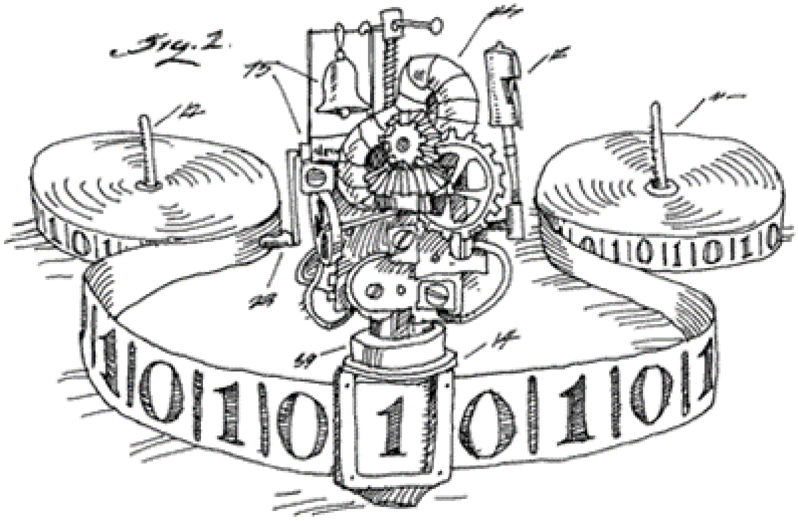
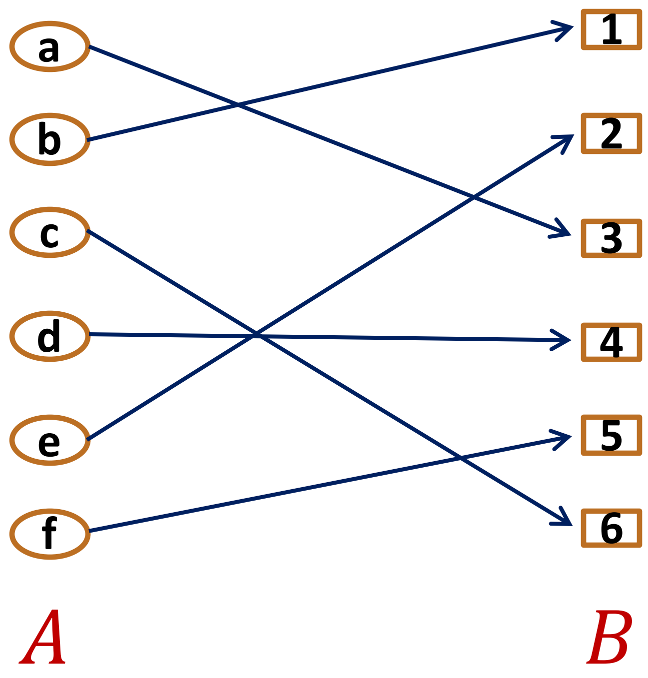
CSE 311: Foundations of Computing

Lecture 29: Cardinality and the Halting problem



Recap: Cardinality

Definition: **Two sets A and B have the same cardinality if there is an bijective (=injective + surjective) function $f : A \to B$.**



Recall that a function $f : A \to B$ is

- Injective, if for for each $y \in B$ there is at most one $x \in A$ with $f(x) = y$
- Surjective, if for every $y \in B$ there is at least one $x \in A$ with $f(x) = y$

Cardinality

Do the natural numbers and the even natural numbers have the same cardinality?

Yes!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	...

What's the map $\boldsymbol{f} : \mathbb{N} \to \mathbf{2}\mathbb{N}$? $\boldsymbol{f(n) = 2n}$

Countable sets

Definition: A set is **countable** iff it has the same cardinality as some subset of $\mathbb{N}$.

Equivalent: A set $\boldsymbol{S}$ is countable iff there is a *surjective* function $\boldsymbol{g} : \mathbb{N} \to \boldsymbol{S}$

Equivalent: A set $\boldsymbol{S}$ is countable iff we can order the elements $\boldsymbol{S = \{x_1, x_2, x_3, \dots\}}$

Example: $\mathbb{Z}$ is countable

Claim: Σ^* is countable for every finite Σ

**Idea: For $k = 0,1,2,\ldots$ list all the $|\Sigma|^k$ many strings of length k.
Then each string in Σ^* appears in that list.**

e.g. {0,1}* **is countable:**

{ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, … }

Countable sets

A set S is **countable** iff we can order the elements of S as

$$S = \{x_1, x_2, x_3, \dots\}$$

Countable sets:

$\mathbb{N}$ - the natural numbers

$\mathbb{Z}$ - the integers

$\mathbb{Q}$ - the rationals

Σ^* - the strings over any finite Σ

The set of all Java programs

The set of all Turing machines

Enumerate in increasing length

Uncountable sets: ???

Lecture 29 Activity

- Please help us improve the quality of this course and take a few minutes to fill out the course evaluation.
- The links are the following.
  CSE 311 A (afternoon): https://uw.iasystem.org/survey/242867
  CSE 311 B (morning): https://uw.iasystem.org/survey/242869

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/philipmg and login with your UW identity

Are the real numbers countable?

Theorem [Cantor]:
The set of real numbers between 0 and 1 is **not** countable.

Proof will be by contradiction.
Uses a new method called diagonalization.

Real numbers between 0 and 1: $[0,1)$

Every number between 0 and 1 has an infinite decimal expansion:

1/2	=	0.5000000000000000000000...
1/3	=	0.3333333333333333333333...
1/7	=	0.14285714285714285714285...
π-3	=	0.14159265358979323846264...
1/5	=	0.1999999999999999999999...
	=	0.2000000000000000000000...

Representation is unique except for the cases that the decimal expansion ends in all 0's or all 9's.
We will never use the all 9's representation.

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

r_1	**0.50000000…**
r_2	**0.33333333…**
r_3	**0.14285714…**
r_4	**0.14159265…**
r_5	**0.12122122…**
r_6	**0.25000000…**
r_7	**0.71828182…**
r_8	**0.61803394…**
…	…

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

		1	2	3	4	5	6	7	8	9	...
r_1	0.	5	0	0	0	0	0	0	0	...	...
r_2	0.	3	3	3	3	3	3	3	3	...	...
r_3	0.	1	4	2	8	5	7	1	4	...	...
r_4	0.	1	4	1	5	9	2	6	5	...	...
r_5	0.	1	2	1	2	2	1	2	2	...	...
r_6	0.	2	5	0	0	0	0	0	0	...	...
r_7	0.	7	1	8	2	8	1	8	2	...	...
r_8	0.	6	1	8	0	3	3	9	4	...	...
...	....	...	....	....	...	...	...	...	...	...	

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

		1	2	3	4	5	6	7	8	9	...
r_1	0.	5	0	0	0	0	0	0	0	...	...
r_2	0.	3	3	3	3	3	3	3	3	...	...
r_3	0.	1	4	2	8	5	7	1	4	...	...
r_4	0.	1	4	1	5	9	2	6	5	...	...
r_5	0.	1	2	1	2	2	1	2	2	...	...
r_6	0.	2	5	0	0	0	0	0	0	...	...
r_7	0.	7	1	8	2	8	1	8	2	...	...
r_8	0.	6	1	8	0	3	3	9	4	...	...
...	....	...	....	....	...	...	...	...	...	...	

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

		1	2	3	4						
r_1	0.	5^{1}	0	0	0						
r_2	0.	3	3^{5}	3	3						
r_3	0.	1	4	2^{5}	8	5	7	1	4	...	...
r_4	0.	1	4	1	5^{1}	9	2	6	5	...	...
r_5	0.	1	2	1	2	2^{5}	1	2	2	...	...
r_6	0.	2	5	0	0	0	0^{5}	0	0	...	...
r_7	0.	7	1	8	2	8	1	8^{5}	2	...	...
r_8	0.	6	1	8	0	3	3	9	4^{5}	...	...
...	....	...	....	....	...	...	...	...	...	...	

Flipping rule:

If digit is 5, make it 1.

If digit is not 5, make it 5.

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

Flipping rule:

If digit is 5, make it 1.

If digit is not 5, make it 5.

		1	2	3	4						
r_1	0.	5^1	0	0	0						
r_2	0.	3	3^5	3	3						
r_3	0.	1	4	2^5	8	5	7	1	4	...	...
r_4	0.	1	4	1	5^1	9	2	6	5	...	...
r_5	0.	1	2	1	2	2^5	1	2	2	...	...
r_6	0.	2	5	0	0	0	0^5	0	0	...	...
r_7	0.	7	1	8	2	8	1	8^5	2	...	...

If diagonal element is $\mathbf{0.}\,x_{11}x_{22}x_{33}x_{44}x_{55}\cdots$ then let's call the flipped number $\mathbf{0.}\,\hat{x}_{11}\hat{x}_{22}\hat{x}_{33}\hat{x}_{44}\hat{x}_{55}\cdots$

It cannot appear anywhere on the list!

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

		1	2	3	4						
r_1	0.	5 (1)	0	0	0						
r_2	0.	3	3 (5)	3	3						
r_3	0.	1	4	2 (5)	8	5	7	1	4	…	…
r_4	0.	1	4	1	5 (1)	9	2	6	5	…	…
						2 (5)	1	2	2	…	…
						0	0 (5)	0	0	…	…
						8	1	8 (5)	2	…	…

Flipping rule:

If digit is 5, make it 1.

If digit is not 5, make it 5.

For every $n \geq 1$:

$r_n \neq 0.\hat{x}_{11}\hat{x}_{22}\hat{x}_{33}\hat{x}_{44}\hat{x}_{55}\cdots$

because the numbers differ on the n-th digit!

If diagonal element is $0.x_{11}x_{22}x_{33}x_{44}x_{55}\cdots$ then let's call the flipped number $0.\hat{x}_{11}\hat{x}_{22}\hat{x}_{33}\hat{x}_{44}\hat{x}_{55}\cdots$

It cannot appear anywhere on the list!

Proof that $[0,1)$ is not countable

Suppose, for the sake of contradiction, that there is a list of them:

		1	2	3	4						
r_1	0.	5 → 1	0	0	0						
r_2	0.	3	3 → 5	3	3						
r_3	0.	1	4	2 → 5	8	5	7	1	4	...	...
r_4	0.	1	4	1	5 → 1	9	2	6	5	...	...
						2 → 5	1	2	2	...	...
						0	0 → 5	0	0	...	...
						8	1	8 → 5	2	...	...

Flipping rule:

If digit is 5, make it 1.

If digit is not 5, make it 5.

For every $n \geq 1$:

$r_n \neq 0.\hat{x}_{11}\hat{x}_{22}\hat{x}_{33}\hat{x}_{44}\hat{x}_{55}\cdots$

because the numbers differ on the n-th digit!

So the list is incomplete, which is a contradiction.

Thus the real numbers between 0 and 1 are **not countable**: “uncountable”

Last time: Countable sets

Countable sets:

$\mathbb{N}$ - the natural numbers

$\mathbb{Z}$ - the integers

$\mathbb{Q}$ - the rationals

Σ^*- the strings over any finite Σ

The set of all Java programs

The set of all Turing machines

} Enumerate in increasing length

Uncountable sets:

$\mathbb{R}$ - the natural numbers

$\mathrm{P}(\mathbb{N})$ - power set of $\mathbb{N}$

Set of functions $f : \mathbb{N} \to \{0,1\}$

Uncomputable functions

We have seen that:

- The set of all (Java) programs is countable
- The set of all functions $f : \mathbb{N} \to \{0,1\}$ is not countable

So: There must be some function $f : \mathbb{N} \to \{0,1\}$ that is not computable by any program!

Interesting… maybe.

Can we come up with an explicit function that is uncomputable?

Some Notation

We’re going to be talking about *Java code*.

CODE(**P**) will mean “the code of the program **P**”

So, consider the following function:

```
public String P(String x) {
    return new String(Arrays.sort(x.toCharArray());
}
```

What is **P**(CODE(**P**))?

“(((()))))..;AACPSSaaabceeggghiiiilnnnnnooprrrrrrrrrrssstttttttuuwxxyy{}”

The Halting Problem

CODE(**P**) means "the code of the program **P**"

The Halting Problem

Given: - CODE(**P**) for any program **P**
- input **x**

Output: **true** if **P** halts on input **x**
false if **P** does not halt on input **x**

Undecidability of the Halting Problem

CODE(**P**) means "the code of the program **P**"

The Halting Problem

Given: - CODE(**P**) for any program **P**
- input **x**

Output: **true** if **P** halts on input **x**
false if **P** does not halt on input **x**

Theorem [Turing]: **There is no program that solves the Halting Problem**

Proof by contradiction

Suppose that H is a Java program that solves the Halting problem.

Proof by contradiction

Suppose that H is a Java program that solves the Halting problem.

Then we can write this program:

```
public static void D(String s) {
    if (H(s,s) == true) {
        ...
    } else {
        ...
    }
}

public static bool H(String s, String x) { ... }
```

Does D(CODE(**D**)) halt?

Does **D**(CODE(**D**)) halt?

```
public static void D(s) {
    if (H(s,s) == true) {
        ...
    }
    else {
        ...
    }
}
```

Does **D**(CODE(**D**)) halt?

```
public static void D(s) {
    if (H(s,s) == true) {
        ...
    }
    else {
        ...
    }
}
```

H solves the halting problem implies that

H(CODE(**D**),s) is **true** iff **D**(s) halts, **H**(CODE(**D**),s) is **false** iff not

Does **D**(CODE(**D**)) halt?

```
public static void D(s) {
    if (H(s,s) == true) {
        while (true); /* don't halt */
    }
    else {
        ...
    }
}
```

H solves the halting problem implies that

H(CODE(**D**),s) is **true** iff **D**(s) halts, **H**(CODE(**D**),s) is **false** iff not

Suppose that **D**(CODE(**D**)) **halts**.

Then, by definition of **H** it must be that

H(CODE(**D**), CODE(**D**)) is **true**

Which by the definition of **D** means **D**(CODE(**D**)) **doesn't halt**

Does D(CODE(D)) halt?

```
public static void D(s) {
    if (H(s,s) == true) {
        while (true); /* don't halt */
    }
    else {
        return;        /*   halt   */
    }
}
```

H solves the halting problem implies that

H(CODE(**D**),s) is **true** iff **D**(s) halts, **H**(CODE(**D**),s) is **false** iff not

Suppose that **D**(CODE(**D**)) **halts.**

Then, by definition of **H** it must be that

H(CODE(**D**), CODE(**D**)) is **true**

Which by the definition of **D** means **D**(CODE(**D**)) **doesn't halt**

Suppose that **D**(CODE(**D**)) **doesn't halt.**

Then, by definition of **H** it must be that

H(CODE(**D**), CODE(**D**)) is **false**

Which by the definition of **D** means **D**(CODE(**D**)) **halts**

Does **D**(CODE(**D**)) halt?

```
public static void D(s) {
    if (H(s,s) == true) {
        while (true); /* don't halt */
    }
    else {
        return;        /*   halt   */
    }
}
```

H solves the halting problem implies that

H(CODE(**D**),s) is **true** iff **D**(s) halts, **H**(CODE(**D**) … ot

Suppose that **D**(CODE(**D**)) **halts.**

Then, by definition of **H** it mu…

H(CODE(**D**), CO…

Which by the defi… **D**(CODE(**D**)) **doesn't halt**

Suppose th… **doesn't halt.**

T… of **H** it must be that

…ODE(**D**), CODE(**D**)) is **false**

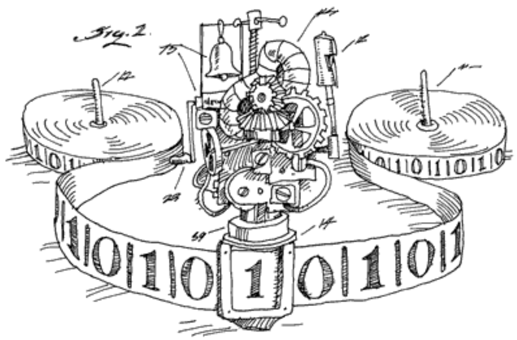
Whi… by the definition of **D** means **D**(CODE(**D**)) **halts**

Contradiction!

The ONLY assumption was that the program H exists so that assumption must have been false.

Done

- We proved that there is no computer program that can solve the Halting Problem.
  – There was nothing special about Java*
    [Church-Turing thesis]



- This tells us that there is no compiler that can check our programs and guarantee to find any infinite loops they might have.

Where did the idea for creating D come from?

```
public static void D(s) {
    if (H(s,s) == true) {
        while (true); /* don't halt */
    }
    else {
        return;         /*    halt    */
    }
}
```

D halts on input code(P) iff H(code(P),code(P)) outputs false
iff P doesn't halt on input code(P)

Connection to diagonalization

Write <P> for CODE(P)

Some possible inputs **x**

All programs **P**	$<P_1>$	$<P_2>$	$<P_3>$	$<P_4>$	$<P_5>$	$<P_6>$	....
P_1							
P_2							
P_3							
P_4							
P_5							
P_6							
P_7							
P_8							
P_9							
.							
.							

This listing of all programs really does exist since the set of all Java programs is countable

The goal of this "diagonal" argument is not to show that the listing is incomplete but rather to show that a "flipped" diagonal element is not in the listing

Connection to diagonalization

Write $<P>$ for CODE(**P**)

Some possible inputs **x**

All programs **P**

	$<P_1>$	$<P_2>$	$<P_3>$	$<P_4>$	$<P_5>$	$<P_6>$	....					
P_1	0	1	1	0	1	1	1	0	0	0	1	...
P_2	1	1	0	1	0	1	1	0	1	1	1	...
P_3	1	0	1	0	0	0	0	0	0	0	1	...
P_4	0	1	1	0	1	0	1	1	0	1	0	...
P_5	0	1	1	1	1	1	1	0	0	0	1	...
P_6	1	1	0	0	0	1	1	0	1	1	1	...
P_7	1	0	1	1	0	0	0	0	0	0	1	...
P_8	0	1	1	1	1	0	1	1	0	1	0	...
P_9	. . . .											
.	. . . .											
.												

(P,x) entry is **1** if program **P** halts on input **x**
and **0** if it runs forever

Connection to diagonalization

Write <**P**> for CODE(**P**)

Some possible inputs **x**

All programs **P**

	<P_1>	<P_2>	<P_3>	<P_4>	<P_5>	<P_6>	….					
P_1	0^{1}	1	1	0	1							
P_2	1	1^{0}	0	1	0							
P_3	1	0	1^{0}	0	0							
P_4	0	1	1	0^{1}	1	0	1	1	0	1	0	…
P_5	0	1	1	1	1^{0}	1	1	0	0	0	1	…
P_6	1	1	0	0	0	1^{0}	1	0	1	1	1	…
P_7	1	0	1	1	0	0	0^{1}	0	0	0	1	…
P_8	0	1	1	1	1	0	1	1^{0}	0	1	0	…
P_9	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.												

Want behavior of program ***D*** to be like the flipped diagonal, so it can't be in the list of all programs.

(P,x) entry is **1** if program **P** halts on input **x** and **0** if it runs forever

Where did the idea for creating D come from?

```
public static void D(s) {
    if (H(s,s) == true) {
        while (true); /* don’t halt */
    }
    else {
        return;        /*   halt   */
    }
}
```

D halts on input code(P) iff H(code(P),code(P)) outputs false
iff P doesn’t halt on input code(P)

Therefore for any program P, D differs from P on input code(P)