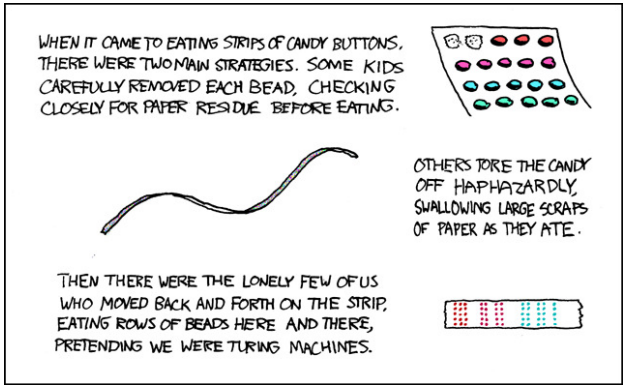
CSE 311: Foundations of Computing

Fall 2013

Lecture 27: Turing machines and decidability



highlights

- Cardinality
- A set S is *countable* iff we can write it as $S=\{s_1, s_2, s_3, \ldots\}$ indexed by $\mathbb{N}$
- Set of rationals is countable
  - "dovetailing"

- Σ^* is countable
  - $\{0,1\}^* = \{\lambda,0,1,00,01,10,11,000,001,010,011,100,101,\ldots\}$
- Set of all (Java) programs is countable

what about the real numbers?

Q: Is *every* set is countable?

A: Theorem [Cantor] The set of real numbers (even just between 0 and 1) is NOT countable

Proof is by contradiction using a new method called diagonalization

proof by contradiction

- Suppose that $\mathbb{R}^{[0,1)}$ is countable
- Then there is some listing of all elements $\mathbb{R}^{[0,1)} = \{ r_1, r_2, r_3, r_4, \ldots \}$
- We will prove that in such a listing there must be at least one missing element which contradicts statement "$\mathbb{R}^{[0,1)}$ is countable"
- The missing element will be found by looking at the decimal expansions of $r_1, r_2, r_3, r_4, \ldots$

real numbers between 0 and 1: $\mathbb{R}^{[0,1)}$

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

$1/2 = 0.500000000000000000000000\ldots$

$1/3 = 0.333333333333333333333333\ldots$

$1/7 = 0.142857142857142857142857\ldots$

$\pi - 3 = 0.141592653589793238462643\ldots$

$1/5 = 0.199999999999999999999999\ldots$

$= 0.200000000000000000000000\ldots$

representations of real numbers as decimals

Representation is unique except for the cases that decimal ends in all 0's or all 9's.

$x = 0.199999999999999999999999\ldots$

$10x = 1.999999999999999999999999\ldots$

$9x = 1.8$ so

$x = 0.200000000000000000000000\ldots$

Won't allow the representations ending in all 9's

All other representations give different elements of $\mathbb{R}^{[0,1)}$

supposed listing of $\mathbb{R}^{[0,1)}$

		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	...	...
...	....	...	....	....	...	...	...	...	...	...	

supposed listing of $\mathbb{R}^{[0,1)}$

		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	...	...
...	....	...	....	....	...	...	...	...	...	...	

flipped diagonal

		1	2	3	4						
r_1	0.	$5^{\bar{1}}$	0	0	0						
r_2	0.	3	$3^{\bar{5}}$	3	3						
r_3	0.	1	4	$2^{\bar{5}}$	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

flipped diagonal number D

		1	2	3	4	5	6	7	8	9	...
D =	0.	1									
			5								
				5							
					1						
						5					
							5				
								5			
									5		
										...	

D is in $\mathbb{R}^{[0,1)}$

But for all **n**, we have **D**≠$\mathbf{r_n}$ since they differ on **n**th digit (which is not **0** or **9**)

⇒ list was incomplete

⇒ $\mathbb{R}^{[0,1)}$ is not countable

why doesn't this show that the rationals aren't countable?

- The set of real numbers is not countable

		1	2	3	4	5	6	7	8	9	...
r_1 D=	0.	5^{1}	0	0	0	0	0	0	0	...	...
r_2	0.	3	3^{5}	3	3	3	3	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}	...	...
...	....	...	....	....	...	...	...	...	...	...	...

flipped diagonal number D

the set of all functions $f : \mathbb{N} \to \{0,1,\ldots,9\}$ is not countable

highlights

- There exist functions that cannot be computed by any program
  - The set of all functions $f : \mathbb{N} \rightarrow \{0,1,\ldots,9\}$ is not countable
  - The set of all (Java/C/C++) programs is countable
  - So there are simply more functions than programs

do we care?

- Are any of these functions, ones that we would actually want to compute?
  - The argument does not even give any example of something that can't be done, it just says that such an example exists
- We haven't used much of anything about what computers (programs or people) can do
  - Once we figure that out, we'll be able to show that some of these functions are really important

before Java...more from our brief history of reasoning

- 1930's
  - How can we formalize what algorithms are possible?

    Turing machines (Turing, Post)

    basis of modern computers

    Lambda Calculus (Church)

    basis for functional programming

    μ-recursive functions (Kleene)

    alternative functional programming basis

    All are equivalent!

turing machines

Church-Turing Thesis

Any reasonable model of computation that includes all possible algorithms is equivalent in power to a Turing machine

- Evidence
  - Intuitive justification
  - Huge numbers of equivalent models to TM's based on radically different ideas

components of Turing's intuitive model of computation

- Finite Control
  - Brain/CPU that has only a finite # of possible "states of mind"
- Recording medium
  - An unlimited supply of blank "scratch paper" on which to write & read symbols, each chosen from a finite set of possibilities
  - Input also supplied on the scratch paper
- Focus of attention
  - Finite control can only focus on a small portion of the recording medium at once
  - Focus of attention can only shift a small amount at a time

what is a Turing machine?

what is a Turing machine?

- Recording medium
  - An infinite read/write "tape" marked off into cells
  - Each cell can store one symbol or be "blank"
  - Tape is initially all blank except a few cells of the tape containing the input string
  - Read/write head can scan one cell of the tape - starts on input
- In each step, a Turing machine
  - Reads the currently scanned symbol
  - Based on state of mind and scanned symbol
    - Overwrites symbol in scanned cell
    - Moves read/write head left or right one cell
    - Changes to a new state
- Each Turing Machine is specified by its finite set of rules

sample Turing machine

	_	0	1
s_1	$(1,s_3)$	$(1,s_2)$	$(0,s_2)$
s_2	(H,s_3)	(R,s_1)	(R,s_1)
s_3	(H,s_3)	(R,s_3)	(R,s_3)

_	_	1	1	0	1	1	_	_

what is a Turing machine?

turing machine ≡ ideal Java/C program

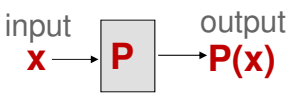
- Ideal C/C++/Java programs
  - Just like the C/C++/Java you're used to programming with, except you never run out of memory
    - constructor methods always succeed
    - **malloc** never fails
- Equivalent to Turing machines except a lot easier to program !
  - Turing machine definition is useful for breaking computation down into simplest steps
  - We only care about high level so we use programs

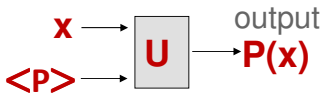
turing's big idea: machines as data

- Original Turing machine definition
  - A different "machine" **M** for each task
  - Each machine **M** is defined by a finite set of possible operations on finite set of symbols
    - **M** has a finite description as a sequence of symbols, its "code"
- You already are used to this idea:
  - We'll write **<P>** for the code of program **P**
  - i.e. **<P>** is the program text as a sequence of ASCII symbols and **P** is what actually executes

turing's idea: a universal turing machine

- A Turing machine interpreter **U**
  - On input **<P>** and its input **x**, **U** outputs the same thing as **P** does on input **x**
  - At each step it decodes which operation **P** would have performed and simulates it.
- One Turing machine is enough
  - Basis for modern stored-program computer
    - Von Neumann studied Turing's UTM design





halting problem

- **Given:** the code of a program **P** and an input **x** for **P**, i.e. given **(<P>,x)**
- **Output:** **1** if **P** halts on input **x**
  0 if **P** does not halt on input **x**

Theorem (Turing): There is no program that solves the halting problem

"The halting problem is undecidable"

proof by contradiction

- Suppose that **H** is a Turing machine that solves the Halting problem

```
Function D(x):
    if H(x,x)=1 then
        while (true); /* loop forever */
    else
        no-op; /* do nothing and halt */
    endif
```

- What does **D** do on input **<D>**?
  - Does it halt?

proof by contradiction

Does **D** halt on input **<D>**?

```
Function D(x):
  • if H(x,x)=1 then
      – while (true); /* loop forever */
  • else
      – no-op; /* do nothing and halt */
  • Endif`
```

D halts on input **<D>**

⇔ **H** outputs **1** on input **(<D>,<D>)**

[since **H** solves the halting problem and so **H(<D>,x)** outputs **1** iff **D** halts on input **x**]

⇔ **D** runs forever on input **<D>**

[since **D** goes into an infinite loop on **x** iff **H(x,x)=1**]

This is contradiction. Our only assumption was that **H** exists. □

that's it!

- We proved that there is no computer program that can solve the Halting Problem.
- This tells us that there is no compiler that can check our programs and guarantee to find any infinite loops they might have