Things computers can’t do!

Aliens invade from CS 5 Gold!
All robots use FSM control

What states can you “factor out” from watching this towel-folding?
Towel-folding states!
Towel-folding?

singled out as a questionable use of dollars...
An autonomous vehicle’s FSM

Fig. 9. Situational Interpreter State Transition Diagram. All modes are sub-modes of the system RUN mode (Fig 4(b)).
FSMs driving robots...

DARPA Urban Challenge 2007
FSMs driving robots...

MIT’s car, Talos - *and its sensor suite*
Final project state machine

- **still broken**
  - stop adding features + start adding print statements

- **it's broken**
  - comment out print statements + start adding more features

- **still works**
  - store a copy somewhere else!

- **it works**
State-machine *limits*?

Are there *limits* to what FSMs can do?

they can’t necessarily *drive safely*...

But are there any *binary-string problems* that FSMs can't solve?
State-machine *limits*?

Let’s build a FSM that accepts strings with **any # of 0s** followed by the **same # of 1s**

- **accepted**
  - 000111
  - 0011
  - 01
  - 00110

- **rejected**
  - 011
  - 001
  - 11100
  - λ (this last string is empty)
Let's build a FSM that accepts strings with **any # of 0s** followed by the **same # of 1s**.

**accepted**
- 000111
- 0011
- 01

**rejected**
- 011
- 001
- 11100
- 00110

0, 1

You don’t need three eyes to see some problems here!
State-machines are limited.

**FSMs can’t count**
at least not arbitrarily high...

We need a more powerful model than FSMs...

What do we need to add?
Turing Machines

A simple model of universal computation

The tape: an unbounded amount of memory. Consists of cells, each containing exactly one character (e.g. 0, 1, or (blank))

The control: a finite amount of memory, the control states — some are accepting, some are not.

Ability to move left and right

The complete state of a TM is determined by:

Starts on the leftmost character

Read/write head for the tape

The symbol now under the head

The symbols to the right of the head

The symbols to the left of the head

The control state
a Turing Machine rule:

an accepting state **always halts** -- then basks in its success!

if a transition is missing, the input FAILS!

"blanks" are the default

0 ; 1 , R

READ WRITE MOTION

0 ; 1 , R
a Turing Machine rule: 0 ; 1 , R

READ WRITE MOTION

an accepting state always halts -- then basks in its success!

if a transition is missing, the input FAILS!

Accepted Input!
a Turing Machine rule:

\[ 0 ; 1 , R \]

READ  WRITE  MOTION

an accepting state **always halts** -- then basks in its success!

if a transition is missing, the input FAILS!

Rejection Input.
Alan Turing
1912-1954

Enigma machine ~ The axis's encryption engine

"Mathematics, rightly viewed, possesses not only truth, but supreme beauty; a beauty cold and austere, like that of sculpture." - Bertrand Russell
The Turing Test

Can machines think?
So far, all known computational devices can compute only what Turing Machines can...

Quantum computation

Molecular computation
http://www.arstechnica.com/reviews/2q00/dna/dna-1.html

Parallel computers

Integrated circuits

Electromechanical computation

Water-based computation

Tinkertoy computation

Turing machine
What inputs are accepted in general?

Extra: How could you change this TM to accept palindromes?
(thought experiment and ex. cr.)
What inputs are accepted in general?

strings with any # of 0s followed by the same # of 1s!

Extra: How could you change this TM to accept palindromes? (thought experiment and ex. cr.)

What does one “loop” of (q0-q1-q2-q3-q0) do?

“eat” the first 0 and last 1

Write a blank to tape

Rejected Input.
allow “unmatched” 0 in middle

“eat” the first and last 0

“eat” the first and last 1

allow “unmatched” 1 in middle
Can TMs compute everything?

Alan Turing says **No!**

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

There are many problems computers can’t solve at all!

Perhaps this is not that surprising…

- CS 5 homework
- rising sea levels
- Picobot (optimal program)
- towel folding (well, fast towel folding…)

What!? (Image of a turtle)
There are well-defined mathematical functions that no computer program or TM can compute even with any amount of memory!

\[ f_B(x) = \begin{cases} 
1 & \text{if } x \text{ is odd} \\
0 & \text{if } x \text{ is even} 
\end{cases} \]

```python
def prog1(x):
    return x % 2
```
functions

These are "int-bool" mathematical functions.

- Input is an integer, $x \geq 0$
- Output is 0/1 (boolean or bit)

\[ f_A(x) = 1 \]

\[ f_B(x) = \begin{cases} 
1 & \text{if } x \text{ is odd} \\
0 & \text{if } x \text{ is even} 
\end{cases} \]

\[ f_C(x) = \begin{cases} 
1 & \text{if } x \text{ is 0, 1, or 2} \\
0 & \text{otherwise} 
\end{cases} \]
Example programs

- Input is one integer, $x \geq 0$
- Output is $0/1$ (boolean or bit)

If programs look different, they are different – even if they compute the same function!

**def prog1(x):**
```python
return x % 2
```

**def prog2(x):**
```python
return x < 3
```

**def prog3(x):**
```python
return 1
```

**def prog4(x):**
```python
return len(str(x+42))>1
```

Let’s match!
functions

\[ f_A(x) = 1 \]

\[ f_B(x) = \begin{cases} 
1 & \text{if } x \text{ is odd} \\
0 & \text{if } x \text{ is even} 
\end{cases} \]

\[ f_C(x) = \begin{cases} 
1 & \text{if } x \text{ is 0, 1, or 2} \\
0 & \text{otherwise} 
\end{cases} \]

\[ f_D(x) = \begin{cases} 
1 & \text{if } x^2 < 8 \\
0 & \text{otherwise} 
\end{cases} \]

---

programs

```
def prog1(x):
    return x % 2
```

def prog2(x):
    return x < 3

def prog3(x):
    return 1

def prog4(x):
    return len(str(x+42))>1

def prog5(x):
    return x in [0,1,2]

def prog6(x):
    if x<2: return x
    else: return prog6(x-2)
```

1. Match each program with the function it computes.
2. How many different functions are on the left side?
3. How many different programs are in the right side?

* Which set is larger: all functions or all programs [int-bool in both cases] - or, are they the same size?

**Worksheet!**

Try on the separate worksheet, use this page for solutions

Name(s): ____________________________
functions

\[ f_A(x) = 1 \]

\[ f_B(x) = \begin{cases} 
1 & \text{if } x \text{ is odd} \\
0 & \text{if } x \text{ is even} 
\end{cases} \]

\[ f_C(x) = \begin{cases} 
1 & \text{if } x \text{ is 0, 1, or 2} \\
0 & \text{otherwise} 
\end{cases} \]

\[ f_C = f_D \]

\[ f_D(x) = \begin{cases} 
1 & \text{if } x^2 < 8 \\
0 & \text{otherwise} 
\end{cases} \]

1. Match each program with the function it computes.
2. How many different functions are on the left side?
3. How many different programs are in the right side?

* Which set is larger: all functions or all programs [int-bool in both cases] - or, are they the same size?

There are many more of these …

programs

\[ \text{def prog1(x):} \]
\[ \quad \text{return } x \% 2 \]

all int inputs: \( x \geq 0 \)

\[ \text{def prog2(x):} \]
\[ \quad \text{return } x < 3 \]

\[ \text{def prog3(x):} \]
\[ \quad \text{return } 1 \]

\[ \text{def prog4(x):} \]
\[ \quad \text{return } \text{len(str(x+42))} > 1 \]

\[ \text{def prog5(x):} \]
\[ \quad \text{return } x \text{ in } [0,1,2] \]

\[ \text{def prog6(x):} \]
\[ \quad \text{if } x < 2: \text{ return } x \]
\[ \quad \text{else: return } \text{prog6}(x-2) \]

There are many more of these … than these!
Uncomputable functions?

There are well-defined mathematical functions that no computer program or TM can compute even with any amount of memory!

Why?

There are many more of these mathematical functions than these programs!

R vs. N
What’s Computation Got to Do, Got to Do With It?

Plan:

- Show that the number of Python programs is *countably infinite* *(a small infinity)*

- Show that the number of possible “computational tasks” is *uncountably infinite* *(a big infinity)*!

Conclusion: ?
functions vs. programs!

the Reals from 0 to 1

functions vs. programs

Positive integers
Programs are “like” integers…

```python
def alien(x):
    if x == 42:
        return True
    else:
        return not \ alien(x+1)
```

`programs = \ N` positive integers

For each program, there is an integer.
For each integer, there is a “program.”

This is true:

- at least a string!
- some have syntax errors...

Every program is a **string**.

Every string is just a sequence of **bits**.

Every sequence of bits is also an **int**!

Programs are integers (and vice-versa)
from **progs** to **ints** ~ and back...

# Python converters from int to program (and back)

```python
def prog( i ):
    """ return the program whose int is i ""
    # convert to a string (just base-128 int!)
    if i <= 0:    return ''
    last_char = chr(i%128)
    return prog(i/128) + last_char

def intify( prog ):
    """ return the int whose program is prog ""
    if prog == '' :    return 0
    last_char = prog[-1]
    return 128*intify(prog[:-1]) + ord(last_char)
```

# to run a string:
# (1) code = compile(p, 'str', 'exec')
# (2) exec(code)
Why use base-128?

**ASCII**
American Standard Code for Information Interchange

encodes 128 specified characters into seven-bit integers (0-127)

---

<table>
<thead>
<tr>
<th>ASCII control characters</th>
<th>ASCII printable characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 NULL (Null character)</td>
<td>32 space</td>
</tr>
<tr>
<td>01 SOH (Start of Header)</td>
<td>33 !</td>
</tr>
<tr>
<td>02 STX (Start of Text)</td>
<td>34 &quot;</td>
</tr>
<tr>
<td>03 ETX (End of Text)</td>
<td>35 #</td>
</tr>
<tr>
<td>04 EOT (End of Trans.)</td>
<td>36 $</td>
</tr>
<tr>
<td>05 ENQ (Enquiry)</td>
<td>37 %</td>
</tr>
<tr>
<td>06 ACK (Acknowledgement)</td>
<td>38 &amp;</td>
</tr>
<tr>
<td>07 BEL (Bell)</td>
<td>39 '</td>
</tr>
<tr>
<td>08 BS (Backspace)</td>
<td>40 (</td>
</tr>
<tr>
<td>09 HT (Horizontal Tab)</td>
<td>41 )</td>
</tr>
<tr>
<td>10 LF (Line feed)</td>
<td>42 *</td>
</tr>
<tr>
<td>11 VT (Vertical Tab)</td>
<td>43 +</td>
</tr>
<tr>
<td>12 FF (Form feed)</td>
<td>44 ,</td>
</tr>
<tr>
<td>13 CR (Carriage return)</td>
<td>45 ;</td>
</tr>
<tr>
<td>14 SO (Shift Out)</td>
<td>46 .</td>
</tr>
<tr>
<td>15 SI (Shift In)</td>
<td>47 /</td>
</tr>
<tr>
<td>16 DLE (Data link escape)</td>
<td>48 0</td>
</tr>
<tr>
<td>17 DC1 (Device control 1)</td>
<td>49 1</td>
</tr>
<tr>
<td>18 DC2 (Device control 2)</td>
<td>50 2</td>
</tr>
<tr>
<td>19 DC3 (Device control 3)</td>
<td>51 3</td>
</tr>
<tr>
<td>20 DC4 (Device control 4)</td>
<td>52 4</td>
</tr>
<tr>
<td>21 NAK (Negative acknowl)</td>
<td>53 5</td>
</tr>
<tr>
<td>22 SYN (Synchronous idle)</td>
<td>54 6</td>
</tr>
<tr>
<td>23 ETB (End of trans. block)</td>
<td>55 7</td>
</tr>
<tr>
<td>24 CAN (Cancel)</td>
<td>56 8</td>
</tr>
<tr>
<td>25 EM (End of medium)</td>
<td>57 9</td>
</tr>
<tr>
<td>26 SUB (Substitute)</td>
<td>58 :</td>
</tr>
<tr>
<td>27 ESC (Escape)</td>
<td>59 ;</td>
</tr>
<tr>
<td>28 FS (File separator)</td>
<td>60 &lt;</td>
</tr>
<tr>
<td>29 GS (Group separator)</td>
<td>61 =</td>
</tr>
<tr>
<td>30 RS (Record separator)</td>
<td>62 &gt;</td>
</tr>
<tr>
<td>31 US (Unit separator)</td>
<td>63 ?</td>
</tr>
<tr>
<td>127 DEL (Delete)</td>
<td>95 ~</td>
</tr>
</tbody>
</table>

functions vs. programs!
int-bool functions are **real numbers**!

(integer predicates)

the Reals from 0 to 1 \[ \mathbb{R} = \text{functions} \]

For each real \#, there is a function.

For each function, there is a real \#.

This is true: **but how?**
for each real # …

any real # from 0-1

one bit at a time...

1 2 3 4 5 6 7 8

\[ r_1 = .11111111 ... \]
\[ r_2 = .10101010 ... \]
\[ r_3 = .11000000 ... \]
\[ r_4 = .01101010 ... \]

there’s a function

IN: a pos. integer x
OUT: a 0/1 bool value

\[ f_1(x) = 1 \]
\[ f_2(x) = \begin{cases} 1 & \text{if } x \text{ is odd} \\ 0 & \text{if } x \text{ is even} \end{cases} \]
\[ f_3(x) = \begin{cases} 1 & \text{if } x \text{ is 1 or 2} \\ 0 & \text{otherwise} \end{cases} \]
\[ f_4(x) = \begin{cases} 1 & \text{if } x \text{ is prime} \\ 0 & \text{otherwise} \end{cases} \]
for each real # …

any real # from 0-1

one bit at a time...

1 2 3 4 5 6 7 8

\[ r_1 = \ldots .11111111 \ldots \]

\[ r_2 = \ldots .10101010 \ldots \]

\[ r_3 = \ldots .11000000 \ldots \]

\[ r_4 = \ldots .01101010 \ldots \]

\[ r_5 = \]

\[ r_6 = \ldots .00000000 \ldots .010 \ldots \]

\[ f_1(x) = 1 \]

\[ f_2(x) = \begin{cases} 1 & \text{if } x \text{ is odd} \\ 0 & \text{if } x \text{ is even} \end{cases} \]

\[ f_3(x) = \begin{cases} 1 & \text{if } x \text{ is 1 or 2} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_4(x) = \begin{cases} 1 & \text{if } x \text{ is prime} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_5(x) = \begin{cases} 1 & \text{if } x \text{ is div. by 3} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_6(x) = \]
for each real # ... any real # from 0-1 one bit at a time...

\[ r_1 = 0.11111111 \ldots \] \[ f_1(x) = 1 \]

\[ r_2 = 0.10101010 \ldots \] \[ f_2(x) = \begin{cases} 1 & \text{if } x \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \]

\[ r_3 = \] For each real number, there's a different function – and vice-versa!

\[ r_4 = 0.01101010 \ldots \] \[ f_3(x) = \begin{cases} 1 & \text{if } x \text{ is 1 or 2} \\ 0 & \text{otherwise} \end{cases} \]

\[ r_5 = 0.00100100 \ldots \] \[ f_4(x) = \begin{cases} 1 & \text{if } x \text{ is prime} \\ 0 & \text{otherwise} \end{cases} \]

\[ r_6 = 0.00000000 \ldots 010 \ldots \] \[ f_6(x) = 0, \text{ except when } x = 42 \]

\[ \text{IN: a pos. integer } x \] \[ \text{OUT: a } 0/1 \text{ bool value} \]
functions vs. programs!

the Reals \( \mathbb{R} \) from 0 to 1 vs. Positive integers \( \mathbb{N} \)

functions \( \checkmark \) vs. programs \( \checkmark \)
To infinity - and beyond

the Reals \( R \)
from 0 to 1

Positive integers \( N \)

functions vs. programs

Uncountably infinite

strictly larger

Countably infinite
Different infinities!?

There are infinitely many functions + programs...

... but not all infinities are created equal!

\{ \text{Cardinal}, \text{Cardinal}, \text{Cardinal} \}

\{ 1, 2, 3 \}

Two sets have equal size if their elements have a one-to-one matching.

This matching is called a bijection.

These sets have the same cardinality.
HOW MATH WORKS:

STEP 1: INSIGHT

MY GOD. I WONDER IF THIS IS TRUE.

STEP 2: RESISTANCE

IMPOSSIBLE! INSANE! IT'S NOT JUST INCORRECT; IT'S AN ENTIRELY NEW CATEGORY OF STUPID!

Thanks, Neel!

No one shall expel us from the paradise that Cantor has created.

Cantor and “friends”

“Scientific Charlatan”
“Renegade”
“Corrupter of Youth”

“Laughable and wrong”
“Utter nonsense”

Leopold Kronecker
1823-1891

Lugwig Wittgenstein
1889-1951

David Hilbert
1862-1943

Ouch!
And this was before 1900!

Lugwig Wittgenstein

"Laughable and wrong"
"Utter nonsense"

"Scientific Charlatan"
"Renegade"
"Corrupter of Youth"
One-to-one mappings also define equal-sized infinite sets, but the results can be surprising!

\[
\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}
\]

Positive integers \( \mathbb{N} = \{1, 2, 3, 4, 5, \ldots\}\)

Positive evens \( \mathbb{E} = \{2, 4, 6, 8, 10, \ldots\}\)

All integers \( \mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}\)

\[
\begin{align*}
\text{def } \text{e2n}(e) & : \\
& \text{return } e//2 \\
\text{def } \text{n2e}(n) & : \\
& \text{return } 2*n \\
\text{def } \text{n2z}(n) & : \\
& \text{return } (-1)**(n\%2) * \backslash \small{(n - (n\%2)) // 2)} \\
\text{def } \text{z2n}(z) & : \\
& \text{return } \text{abs}(z)*2 + (z<0)
\end{align*}
\]

\( \mathbb{E} \) and \( \mathbb{N} \) and \( \mathbb{Z} \) all have the same size!

These sets are said to be countably infinite.

Extra credit: Prove that the set of all rationals \( \mathbb{Q} \) is also countably infinite.
Let's get *real*

the reals  
(from 0 to 1)  
{ 0.0 ≤ r < 1.0 }

Are these sets equal-sized?

the integers  
(positive)  
{ 1, 2, 3, 4, 5, … }

Why is this **NOT** a valid matching? (multiple flaws)

...
There are always real #s missing from any list!

The Reals from 0 to 1

real #s are always missing

regardless of the matching...

\[ r_0 = \]

\[ r_1 = .33333333333333333333333333333333... \] \[ \leftarrow 1 \]

\[ r_2 = .42424242424242424242424242424242... \] \[ \leftarrow 2 \]

\[ r_3 = .314159426535897... \] \[ \leftarrow 3 \]

\[ r_4 = .0909090909090909090909090... \] \[ \leftarrow 4 \]

\[ r_5 = ... \] \[ \leftarrow 5 \]
Cantor Diagonalization

There are always real #s missing from any list!

\[ r_0 = .4350... \]
\[ r_1 = .333333333333333... \]
\[ r_2 = .424242424242424... \]
\[ r_3 = .314159426535897... \]
\[ r_4 = .090909090909090... \]
\[ r_5 = ... \]

\[ r_0[i] = (r_1[i] + 1) \% 10 \]

\[ r_0 \neq r_1 \neq r_2 \neq r_3 \neq r_4 \]

Proving that \([0,1)\) is uncountably infinite!
A Bag of Reals

I’m going to reach into this bag of real numbers and pick one out!

Wow, that’s a fun game!
How Math Works:

Step 1: Insight

My God. I wonder if this is true.

Step 2: Resistance

Impossible! Insane! It's no. It's cute.

Step 6: Transmission to Students.

How do you not get this concept? We spent an hour on it yesterday.

Check the last panel ...
(Many) more **functions** than **programs**!

- **functions**
  - **Uncountably infinite**
- **programs**
  - **Countably infinite**

The Reals from 0 to 1 \( \mathbb{R} \)
Positive integers \( \mathbb{N} \)
There are *lots of* functions

*uncountably* many

Well-defined mathematical functions
(countable)

Programs
(countable)

There are *“a few”* programs
*countably* many

What's swimming around out here?
Philosophically thinking …

How big is the set of the human-describable things… ?

Well-defined mathematical functions

Human-describable things… finite!

… and how does it overlap with all functions or all programs?
Do all human-describable things include all functions? (No...)

Well-defined mathematical functions

$\mathbb{R}$

Programs

$\mathbb{N}$

Human-describable things...

finite!
Well-defined mathematical functions

Programs

Human-describable things… finite!

Do all HD things include all programs? (No...)
It overlaps incompletely – *but also goes beyond...*!

**Well-defined mathematical functions**

- **Wow! Things out here are *indescribable***

- **So, what *is* all this stuff out here?!?**

**Next time!**

- **Human-describable things...**
  - finite!

**Programs**

- **N**

- **R**