Universality of Computation

What is computing?
Physical Computation

ChAOS Optical Layout

Science Camera

Beam Splitters

Lensed Array

Wavefront Sensor

Interferometer

Off-Axis Parabola #1
(Collimator)

Off-Axis Parabola #2
(Camera mirror)

He-Ne Laser

Tip/Tilt Mirror

Tod Mires

Light from Telescope

(use laser out of plane of this page, reflects off of a mirror (not shown) and is sent to the fold Mirrors)
Mathematically defined computation

\[
\begin{align*}
    f(x, 0) &= 1 \\
    f(x, y) &= g(y, f(x, y-1)) \\
    g(z, 0) &= 0 \\
    g(z, w) &= h(w, g(z, w-1)) \\
    h(r, 0) &= r \\
    h(r, s) &= h(r, s-1) + 1
\end{align*}
\]

```python
def is_prime(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```
Turing Machines

Symbol read

<table>
<thead>
<tr>
<th>state</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0, R, b</td>
<td>1, L, g</td>
</tr>
<tr>
<td>b</td>
<td>1, L, b</td>
<td>0, L, a</td>
</tr>
<tr>
<td>z</td>
<td>1, R, x</td>
<td>0, L, a</td>
</tr>
</tbody>
</table>

Symbol written | Head Movement | New State
### Turing Machine for adding 1

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1, _, b</td>
<td>0, L, a</td>
</tr>
<tr>
<td>b</td>
<td>0, _, b</td>
<td>1, _, b</td>
</tr>
</tbody>
</table>

**Start** (with head on least significant bit)  
**Halt**
Universal Computers

• Theoretical version: there exists a single Universal Turing Machine that can simulate all other Turing machines
  – The input will be a coding of another machine + an input to the other machine
• Hardware version: stored program computer
• Software version: interpreter

```python
def simulate(program, input):
    """ Accepts a string that holds a python function definition and an input string, and simulates the function when it is given the input string as its parameter, returning the value that the function returns""
    ...
```
The Church-Turing Hypothesis

Everything “computable” is computable by a Turing machine

– Physical interpretation
– Conceptual interpretation
– Definition of Computation
– Everything computable is computable by a Python program
– All “good enough” computers are equivalent
CTH: The Modern Version

Everything efficiently computable is computable efficiently by a randomized Turing machine

– “Efficiently”: in “polynomial time”. Running time is $O(n^c)$ for some constant $c$.

– “Randomized”: allow to “flip coins” – use randomization

– All “good enough” computers are equivalent -- even if you worry about efficiency
CTH and Quantum Physics

- Only known challenge to the “modern” version of CTH
- We do not know how to simulate quantum mechanical systems efficiently
- The quantum “probability amplitudes” are different from normal probabilities
- Maybe “quantum computers” can efficiently do more than normal randomized computers
  - Efficient “quantum algorithm” for factoring is known
- Can “quantum computers” be built?
  - Existing physics? New physical laws? Technology?
- “Quantum CTH version”: .. By a quantum Turing Machine
CTH and Chaos

• The physical world is analog
• When we simulate it on a digital device we use a given precision
  – Input and output are given in finite precision
  – How many bits of precision do we need?
  – How much can $\varepsilon$ error in the input affect the output?
  – Normally: a little
  – Chaotic systems: a lot
• When we simulate a digital device on an analog device we can encode many bits in one analog signal
  – Limits to precision of analog systems: atoms, quantum effects
  – Hard to think of a physical signal of even 64 bits of precision
CTH and the Brain

• Is the brain just a computer?
• Metaphysics: The mind-body problem
  – Monism: everything is matter -- including humans
  – Dualism: there is more ("soul", "mind", "consciousness" …)
• Technical differences
  – Parallel, complicated, analog, …
  – Still equivalent to a Turing Machine
AI

• Why can’t we program computers to do what humans easily do?
  – Recognize faces social situations
  – Understand human language (well)

• Processing power?

• Software?
  – Scruffy vs. Neat debate

• Big data
The Turing Test

A

The Turing Test

B

Which is conscious: A or B?
The Brain vs. Google Cloud

• There are $10^9$ -- $10^{10}$ neurons in the brain
• Each neuron receives input from $10^3$ -- $10^4$ synapses
• Each neuron can fire about $10^0$ -- $10^2$ times/sec
• Total computing power of brain is $10^{12}$ -- $10^{16}$ ops/sec
• Google cloud has $10^7$--$10^9$ cores
• Each core can compute $10^8$--$10^{10}$ ops/sec
• Total computing power of Google cloud is $10^{15}$ -- $10^{19}$ ops/sec
When Computers are Smarter than us?

• Can do “intelligent” things that we care about better than humans.
  – Scientific/Medical Discoveries, Winning/Avoiding Wars, Running Companies/Commerce, Creating Movies/Books/Games, Educating, Designing Computers/Programs/Systems/Robots...

• Will never happen?
• Quite useful, but no big deal?
• Singularity?
• Augmented-men vs. Huddled masses?
• Robot Uprising?
The limits of computation and mathematics
Universal Computers

• Theoretical version: there exists a single Universal Turing Machine that can simulate all other Turing machines
  – The input will be a coding of another machine + an input to the other machine

• Hardware version: stored program computer

• Software version: interpreter

```python
def simulate(program, input):
    """ Accepts a string that holds a python function definition and an input string, and simulates the function when it is given the input string as its parameter, returning the value that the function returns"""
```
The Halting problem

def halt(program, input):
    """ Accepts a string that holds a python function definition and an input string, and simulates the function when it is given the input string as its parameter, returning True if the program ever returns and returning False if the program never terminates and never returns(e.g. goes into an infinite loop)"""

def auto_halt(program):
    """ returns true if the given program halts when given itself as an input"""

    return halt(program, program)

def paradox(program):
    if autoHalt(program):
        while(True):
            pass
    else
        return True;
The halting problem can not be solved!

Assume: There is a program for computing $\text{halt}$

$\Rightarrow$ There are also programs for $\text{self\_halt}$ and $\text{paradox}$

Question: Will $\text{paradox}$ halt when given its own code as its parameter?

Yes? $\Rightarrow$ $\text{auto\_halt}$ should return True when given $\text{paradox}$’s code as input $\Rightarrow$ $\text{paradox}$ will go into an infinite loop $\Rightarrow$ $\text{paradox}$ will not halt $\Rightarrow$ contradiction

No? $\Rightarrow$ $\text{auto\_halt}$ should return False when given $\text{paradox}$’s code as input $\Rightarrow$ $\text{paradox}$ will return immediately $\Rightarrow$ contradiction

Contradiction to the assumption!

Theorem: the halt method can not be written.
Diophantine Equations cannot be Solved

**Fact:** It is possible to write 
`compile_to_equations`

**Proof idea:** a number can encode the 
complete history of the 
computation.

**Theorem:** There does not exist any 
program that can solve 
diophantine equations

\[
\begin{align*}
x^3 y^2 z + xwz - 3 &= 0 \\
x^2 z + 3w^2 z^3 - 2z &= 0 \\
\ldots \\
z^7 w - zw^6 - 2 &= 0
\end{align*}
\]

```python
def compile_to_equations(program, input):
    """ Accepts a string that holds a python function definition and an input string, and returns a set of Diophantine equations that have a solution if and only if the program halts on the given input."""
```
Formal Proofs

Theorem: \( \Omega \vDash \exists M, \exists M \vdash O \vdash \phi \phi \phi \)

Proof:
\[
\begin{align*}
\Omega &\vDash \bullet \circ M \vDash \rho \rho \\
\exists M, \Omega M \vDash \rho \rho M \\
\exists \exists \Omega \bullet \circ \rho \vDash \bullet \bullet \\
\cdots \\
\end{align*}
\]

Axiom

Follows from previous lines

\( \Omega \vDash \exists M, \exists M \vdash O \vdash \phi \phi \phi \)

QED
Axiomatic System

• A *syntactic* way to show semantic mathematical truths
• Axioms + Deduction rules (logic)
• Must be *effective*
  – trivial to determine if a proof is correct
• Must be *sound*
  – Anything you prove must be true
• Can it be *complete*?
  – Prove all true mathematical statements
Zermelo-Fraenkel Set Theory

• The basis of all mathematics (one possibility)
  \[ \forall S \forall T [ \forall z (z \in S \iff z \in T) \implies S = T] \]
  \[ \forall S \exists T \forall z (z \subseteq S \iff z \in T) \]

...

• Effective
• Sound, we think
• Anything you prove in Math courses really uses ZFC
• The details are not important, yet we will prove:
• **Theorem:** ZFC is not complete
def is_zfc_proof(theorem, proof):
    """ Accepts a string that holds a supposed theorem and a string that holds a supposed ZFC proof of the theorem, and returns True if this is indeed a valid ZFC proof of the theorem."""
Computation theory is part of mathematics

```python
def compile_to_halt_thm(program, input):
    """ Accepts a string that holds a python function definition and an input string, and returns a string of a supposed theorem that states that the program halts on the given input."""

def compile_to_nohalt_thm(program, input):
    """ Accepts a string that holds a python function definition and an input string, and returns a string of a supposed theorem that states that the program does not halt on the given input."""
```
Godel’s incompleteness theorem

\[\text{Theorem: } \text{ZFC is not complete (and neither is anything else)}\]

\[\text{def } \text{halt}(\text{prog, inp}):\]
\[\text{thm}_y = \text{compile_to_halt_thm(} \text{prog, inp)}\]
\[\text{thm}_n = \text{compile_to_nohalt_thm(} \text{prog, inp)}\]
\[\text{for } p \text{ in all_possible_strings:}\]
\[\text{if is_zfc_proof(} \text{thm}_y, p):\]
\[\text{return True}\]
\[\text{if is_zfc_proof(} \text{thm}_n, p):\]
\[\text{return False}\]

\[\text{if all true mathematical statements have ZFC proofs then } \text{halt} \text{ can be solved!}\]