What Are Computers, Really?

Clarissa Littler

July 1, 2014

Outline

What Are Computers, Really?

- Physical devices that carry out computation

What Are Computers, Really?

- Physical devices that carry out computation
- We're done!

Questions?

Thank you all you've been a wonderful audience

What Is Computation, Really?

- Defined computers via computation
- What's computation?

What Is Computation, Really?

- Defined computers via computation
- What's computation?
- It's what computers do, OBVI!

What is Computation, Really for Totes Realsies This Time?

- Circular definitions are bad
- Define computation independent of physical machines
- Computation $\subseteq$ Mathematics

What is Computation, Really for Totes Realsies This Time?

- Circular definitions are bad
- Define computation independent of physical machines
- Computation $\subseteq$ Mathematics
- Computation $\subset$ Mathematics

Why Do We Care?

- Computers have limits
- Limits are independent of hardware
- Chained by the laws of physics and mathematics

Limits of Computation

- No perfect virus scanner

Limits of Computation

- No perfect virus scanner
- No program to perfectly check for infinite loops

Limits of Computation

- No perfect virus scanner
- No program to perfectly check for infinite loops
- No program to calculate proofs of all true theorems

Limits of Computation

- No perfect virus scanner
- No program to perfectly check for infinite loops
- No program to calculate proofs of all true theorems
- Wait, what?

Computation: Examples

- Database queries

Computation: Examples

- Database queries
- Counting

Computation: Examples

- Database queries
- Counting
- Serving web pages

Computation: Examples

- Database queries
- Counting
- Serving web pages
- What do these have in common?

Computation: An Informal Definition

- Computable functions:
  - finite time
  - finite rules
  - finite data

Computation: An Informal Definition

- Computable functions:
  - finite time
  - finite rules
  - finite data
- What about operating systems and servers?

Computation vs. Mathematics

- Computable functions capture notion of algorithm

Computation vs. Mathematics

- Computable functions capture notion of algorithm
- Roughly a function is computable when there exists a program for it

Computation vs. Mathematics

- Computable functions capture notion of algorithm
- Roughly a function is computable when there exists a program for it
- Cover exact definition later

Computation vs. Mathematics

- Computable functions capture notion of algorithm
- Roughly a function is computable when there exists a program for it
- Cover exact definition later
- Are all mathematically definable functions computable?

Computation vs. Mathematics

- Computable functions capture notion of algorithm
- Roughly a function is computable when there exists a program for it
- Cover exact definition later
- Are all mathematically definable functions computable?
- Answer: Not even remotely!

What Are Sets?

- Sets are collections of "things"

What Are Sets?

- Sets are collections of "things"
- "Things" can be any definable concept

What Are Sets?

- Sets are collections of "things"
- "Things" can be any definable concept
  - Set of all cat pictures on the internet

What Are Sets?

- Sets are collections of "things"
- "Things" can be any definable concept
  - Set of all cat pictures on the internet
  - Set of all burritos eaten from food carts to this date

What Are Sets?

- Sets are collections of "things"
- "Things" can be any definable concept
  - Set of all cat pictures on the internet
  - Set of all burritos eaten from food carts to this date
  - Set of all talks you'd rather be at *right now*

Size of Sets

- Finite sets are sets you can count up to a finite size
- The number you reach when counting elements is the *cardinality*
- We write cardinality of a set A as $|A|$
- The set of counting numbers itself a set: $\mathbb{N}$
- Can you count $\mathbb{N}$ to a finite number?

Size of Sets

- Finite sets are sets you can count up to a finite size
- The number you reach when counting elements is the *cardinality*
- We write cardinality of a set A as $|A|$
- The set of counting numbers itself a set: $\mathbb{N}$
- Can you count $\mathbb{N}$ to a finite number?
- No! A logical contradiction otherwise.

Countably infinite

- Sets that are still countable, but the counting doesn't end in finite time
- $\mathbb{N}$ is countable: cardinality $\aleph_0$
- The rational numbers are countable
- The integers are countable

Countably infinite

- Sets that are still countable, but the counting doesn't end in finite time
- $\mathbb{N}$ is countable: cardinality $\aleph_0$
- The rational numbers are countable
- The integers are countable
- The number of finite strings definable over a finite alphabet is countable
  - Goedel numberings

Uncountably infinite

- Sets of functions: $|A \to B| = |B|^{|A|}$
- Set of functions $|\mathbb{N} \to \mathbb{N}| = \aleph_0^{\aleph_0}$

Uncountably infinite

- Sets of functions: $|A \to B| = |B|^{|A|}$
- Set of functions $|\mathbb{N} \to \mathbb{N}| = \aleph_0^{\aleph_0}$
- This is not countable

Uncountably infinite

- Sets of functions: $|A \to B| = |B|^{|A|}$
- Set of functions $|\mathbb{N} \to \mathbb{N}| = \aleph_0^{\aleph_0}$
- This is not countable
- This isn't even remotely countable

Uncountably infinite

- Sets of functions: $|A \to B| = |B|^{|A|}$
- Set of functions $|\mathbb{N} \to \mathbb{N}| = \aleph_0^{\aleph_0}$
- This is not countable
- This isn't even remotely countable
- This is larger than you can possibly wrap your head around

What's The Point?

- Programs are finite strings over a finite alphabet

What's The Point?

- Programs are finite strings over a finite alphabet
- The set of programs is $\aleph_0$

What's The Point?

- Programs are finite strings over a finite alphabet
- The set of programs is $\aleph_0$
- The set of computable functions is the size of the set of programs

What's The Point?

- Programs are finite strings over a finite alphabet
- The set of programs is $\aleph_0$
- The set of computable functions is the size of the set of programs
- The set of mathematically definable functions is larger than $\aleph_0$

What's The Point?

- Programs are finite strings over a finite alphabet
- The set of programs is $\aleph_0$
- The set of computable functions is the size of the set of programs
- The set of mathematically definable functions is larger than $\aleph_0$
- The set of computable functions is vanishingly small compared to all functions

What Are The Implications?

- There's a reason programming is hard
- Can't even imagine a world in which all functions are computable
  - Database queries
  - Search results
  - Cryptography
- Physics and speed of information?

Turing's Insight

- Computer was a title not an object
- Turing watched how computers work
- Think elementary school arithmetic worksheets:
  - Problems *are* scratch paper
  - Clearly defined order in which digits are added
  - Can stop and come back to problem

Turing Machines

- A universal model of computation!
- Simplifies scratch paper: 1d not 2d
- Always uses a finite amount of scratch paper
- Has only a finite number of rules
  - can look at input and remember a finite number of things

The Use of Turing Machines

- Universal Turing Machine
- Textual descriptions of Turing Machines are "code"
- Ties back into cardinality argument
- Dicuss difficulty of problems: decideable vs. recognizable

Decideable

- Decideable means the problem can always be solved in finite time
- Decideable problems:
  - RegExp matching
  - Arithmetic
  - Sorting
  - Checking if a proof is valid
  - (Most) type-checking

Recognizable

- Recognizable means well-formed inputs can be solved in finite time
  - i.e. the input has a proper solution and is written in the right format
- Recognizable problems:
  - Mostly meta-properties or meta-programming: Rice's theorem
  - Termination on a given input
  - Testing a program's behavior
  - Some type-checking (i.e. Scala)
  - C++ templates

Why Are Some Problems Not Decideable?

- Logical paradox
- Halting problem: does a program halt on a given input?
- Variant on Russels's paradox
  - Set of all sets that don't contain themselves
  - Program that terminates only if a program doesn't terminate on itself
- Once you prove one thing isn't decideable, use it to prove others

Untyped Lambda Calculus

- A different model of computation
- Invented by Alonzo Church as foundation of mathematics
- Works as a universal model of computation!
- Theory of function abstraction and application
- `function (x) {body}`
- Variables, anonymous functions, function calls

Church-Turing Thesis

- Lambda calculus equivalent to Turing Machines
- Church-Turing thesis says there is no stronger model of computation
- Evidence: every universal model we know can simulate each other
- Still just a hypothesis! Not proven, but very likely

Turing Completeness and You

- Turing complete means a PL is equivalent to Turing machines
- Most languages are
- This is why you can accidentally make infinite loops

Typed Lambda Calculus

- Lambda calculus with *types*
- Types are stronger than you're used to
- Well-typed programs can't loop
- Not Turing complete

Curry-Howard

- Curry-Howard correspondence
- Types are theorems
- A program of type A is a proof that the theorem A is true
- $A \rightarrow A$ means A implies A
- In Agda
  ```
  (A : Set) -> (a b c : A) -> a == b -> b == c -> a == c
  ```
  is a type that expresses transitivity of equality

Take-aways

- Programming is a subset of mathematics
- Understanding the limits of computation is useful
- Types can be more than bug catchers, they're theorems we can prove