Meeting 1: What Is Computation?

Philosophy of Computation at Berkeley
pocab.org

April 21, 2017

1 Ada Lovelace, the Poetical Scientist

Ada Lovelace, born in 1815, was arguably the first computer scientist, though she didn't call herself that. Instead, she christened herself the "poetical scientist". She was the first person to understand the distinction between a computer and a calculator. That is, while computers churn numbers, the numbers they churn may represent something other than numbers, such as music, poetry, (or even) intelligence. She died of cancer at the age of 36. At her deathbed, she mused, *[I will have] the most* harmoniously *disciplined troops; consisting of vast* numbers*, and marching in irresistible power to the sound of* Music*. Is not this very mysterious?...But then, what* are *these Numbers? There is a riddle* –

When she was not in her deathbed, she also said these things:

> I do not believe that my father was such a poet as I shall be an analyst; for with me the two go together indisputably.
>
> [The Analytical Engine][1] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine...Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.
>
> We may say most aptly, that the Analytical Engine *weaves algebraical patterns* just as the Jacquard-loom weaves flowers and leaves.

A century and a half since her death, these ideas remain radical and unnoticed by almost everyone. In the mid-nineteenth centry, where not a single computer existed, they were revolutionary.

> To fully appreciate her contribution, we would need to look at it from a historical perspective. People at that time thought that romanticism (imagination or the "left brain") and rationalism (reasoning or the "right brain") were polar opposites.
> –Aaron Lai

- It seems that this dichotomy between romanticism and rationalism persists today. (1) Why do you think the dichotomy exists, (2) do you think the dichotomy necessarily exists and so will continue to exist for no less than a million more years, and (3) do you think the dichotomy ought to exist?
- Recall our discussion on abstraction levels of computation. Consider the following two scenarios:
  - I compute $3 + 3 = 6$ using a piece of paper and a pen.
  - I compute $3 + 3 = 6$ by having someone physically reach into my brain and wire my neurons in such a way that two layers of neurons, each encoding 3 (11_2), undergo some computation and output a third layer of neurons encoding 6 (110_2)

[1] the first computer by the eccentric visionary Charles Babbage, never fully finished

I probably use more neurons in scenario 1 than in scenario 2. Thus we could say that scenario 1's computation is at a higher level of abstraction than scenario 2's. When Lovelace says the "engine might compose elaborate and scientific pieces of music", what abstraction level do you suppose she is on?

- Back in the days, it was considered common sense, an obvious fact, an unquestioned assumption, that any educationally enlightened, and thus moral, person should be able to compose beautiful poetry. In fact, one of the biggest qualifications for the Chinese and Korean Civil Service Exam was to compose poetry. Do you think one must be moral to be good at computer science? Similarly, do you think one must be moral to compose good poetry? Do you think there is any inherent difference?

- Take your favorite algorithm, such as mergesort, and write a poem about it no less than five lines with a ABCAC rhyming scheme, using only the words in the quotations of Ada Lovelace in this paper.

2 Turing Machines and the Halting Problem

> Two mathematicians are discussing a theorem. The first mathematician says that the theorem is trivial. In response to the others request for an explanation, he then proceeds with two hours of exposition. At the end of the explanation, the second mathematician agrees that the theorem is trivial.
> – On "trivial" in mathematics, Shreevatsa

- When was the first time you heard the word "trivial" in a mathematical context? How did it come off as? Smug? Enlightened? Beautiful? No opinion?

There are many expositions of Turing machines, what they are, what they're supposed to do, and what they mean. Most of them rely on heavy formal notation. Here is an attempt at an intuitive, and not any less precise, explanation. The world, or any part of the world, can be thought of as a series of *chunks*[2]. Some of these series are Turing machines; (perhaps) others are not. A Turing machine is a *special* series of *chunks* in the world. If you look at any two of these chunks where the second chunk is right after the first chunk, you can find an exact, specifiable, point-finger-at-able reason *why* the second chunk follows from the first chunk.[3] In fact, that is also a definition of *computation*: you can point at some *rule*, agreed upon beforehand, on why the next chunk follows from the previous chunk.[4] This definition may clairfy why a computation is equivalent to a mathematical proof: a proof consists of a sequence of statements, each statement following from the previous statement because of some rule (or, at the very beginning of the proof, some rock-bottom axiom).

In his lecture, Scott Aaronson (Berkeley Ph.D. under Vazirani) discusses the Universal Turing machine, which is a Turing machine capable of computing any Turing machine.

> Turing's first result is the existence of a "universal" machine: a machine whose job is to simulate any other machine described via symbols on the tape. In other words, universal programmable computers can exist. You don't have to build one machine for email, another for playing DVD's, another for Tomb Raider, and so on: you can build a single machine that simulates any of the other machines, by running different programs stored in memory. This result is actually a lemma, which Turing uses to prove his "real" result.
>
> So what's the real result? It's that there's a basic problem, called the halting problem, that no program can ever solve. The halting problem is this: we're given a program, and we want to decide if it ever halts. Of course we can run the program for a while, but what if the program hasn't halted after a million years? At what point should we give up?
> – "Gödel, Turing, and Friends", Scott Aaronson

[2] Chunks of what? Information.

[3] A catch: this explanation requires us to assume that the Church-Turing thesis is true. Perhaps an interesting meta-point: an assumption drastically simplifies an explanation building on that assumption.

[4] An astonishing fact is that the set of previously-agreed-upon-rules might not really matter, that at a certain degree of complexity, they are all the same.

- Do you think humans are universal Turing machines? In other words, is a human "nothing but" a machine capable of executing any Turing machine whatsoever? Is the sentiment in "nothing but" in the previous sentence justified?
- An objection might be that humans cannot compute for an infinite amount of time, while a Turing machine (theoretically) can. Is there a way out of this objection, or not?[5]

Craig Kaplan, professor at University of Waterloo, provides one of the best explanations on the halting problem I have ever read. He has graciously consented to the following excerpt existing on our worksheet.

```
bool bobs_yer_uncle( char * program ) {
     if( stops_on_self( program ) ) {
        while( 1 ) {}
        return FALSE;
    } else {
        return TRUE;
    }
 }
```

> Here's the paradox. What happens when I try `bobs_yer_uncle` on itself? Well, clearly one of two things can happen: either it runs forever, or it stops and returns TRUE, depending on whether the call to `stops_on_self` returns TRUE or FALSE.
>
> If `bobs_yer_uncle( bobs_yer_uncle )` goes into an infinite loop, it is because `stops_on_self( bobs_yer_uncle )` returned TRUE, which means that `would_it_stop( bobs_yer_uncle, bobs_yer_uncle )` returned TRUE. But this means that `bobs_yer_uncle` would stop when fed itself as input! This contradicts the assumption that it goes into an infinite loop. If `bobs_yer_uncle( bobs_yer_uncle )` stops and returns TRUE, it's because `stops_on_self( bobs_yer_uncle )` returned FALSE, which means that would_it_stop(`bobs_yer_uncle`, `bobs_yer_uncle`) returned FALSE. But this means that `bobs_yer_uncle` would run forever when fed itself as input! This contradicts the assumption that it terminates.
> – Understanding the Halting Problem

- What's the big fuss, if any, about this problem?
- What is the core cleverness, if any, in this problem? Can you give the cleverness a name?
- Can a human "solve" the halting problem in any meaningful sense of the word?
- One might say (as Hofstadter does) that one crucial fact that separates humans from animals is that humans are self-referential, that is, they can use self-reference to modify themselves. Do you (1) agree humans are separate from animals? (2) If so, does self-reference have anything to do with it? If so, (3) self-reference gives humans a certain amount of "power", at least enough to distinguish us from animals. How can self-reference be so magical?

[5] *Where Mathematics Comes From*, George Lakoff, discusses how the human mind actually thinks of infinity.

3 The Church-Turing Thesis

> The Church-Turing thesis says that any real-world computation can be translated into an equivalent computation involving a Turing machine.
> –Wolfram MathWorld

The Church-Turing thesis is really the Church-Turing *hypo*thesis, because it hasn't ever been "proved", though it has never been disproved[6]. It is probably an empirical question, not a mathematical one. That does not mean much to detract from its significance, because, arguably, rationality requires faith in rationality to begin with[7].

Are you a believer? If so, devise a cult to spread this idea to many people. What would be your sales pitch?

4 Child Development and Computation

Jean Piaget is perhaps the most famous developmental psychologist. He wrote a lot of books about how babies grow into children grow into adults. His most famous explanation of development is that development is a constant tug-of-war between *assimilation* and *accommodation*. To make the terms precise, assimilation is "the process by which the individual deals with an environmental event in terms of current structures", while accommodation is "the individuals tendency to change in response to environmental demands" (Ginsburg & Opper 1988).

Alternative explanations:

> In Assimilation, what is perceived in the outside world is incorporated into the internal world, without changing the structure of that internal world, but potentially at the cost of "squeezing" the external perceptions to fit hence pigeon-holing and stereotyping.
>
> In Accommodation, the internal world has to accommodate itself to the evidence with which it is confronted and thus adapt to it, which can be a more difficult and painful process.
>
> In reality, both are going on at the same time, so that – just as the mower blade cuts the grass, the grass gradually blunts the blade – although most of the time we are assimilating familiar material in the world around us, nevertheless, our minds are also having to adjust to accommodate it.
>
> – Atherton J S

- Consider the following interpretation of assimilation and accommodation using ideas from computation: assimilation is using an algorithm to operate on an outside thing, while accommodation is modifying a pre-existing algorithm or *inventing* a new one. Is this a valid interpretation, or is it too reductionist? If valid, does accommodation have anything to do with self-reference? If so, are the following processes analogous? At a surface level, or at a deeper level?
  - Self-reference is necessary for uncomputability.
  - Accommodation is necessary for invention.

[6] Quantum computation threatens its position a little, but probably not too much.

[7] Hume's problem of induction: before we can say that A implies B, we must be able to say that such-and-such a case implies *implication*. To trust implication, we must first trust in implication.