Computational Thought

Professor Dennis Shasha
Scribe: Justin de Guzman

Week 1

September 9, 2013

Introduction

Professor Shasha introduced the class first with the course description and an overview of the syllabus. He stated that Isaac Newton did not describe himself as a physicist, but instead a natural *philosopher*. This class will cover the **philosophy of mathematics**.

View Syllabus

Required Texts

- *Natural Computing: DNA, Quantum Bits, and the Future of Smart Machines*
- *The Puzzling Adventures of Dr. Ecco*

Scribes

Each student will be responsible as a scribe for two half-classes. See the following links for examples:

Example 1
Example 2
Example 3

Presentation

Each student is responsible for a presentation about the scientist discussed in that week's current chapter. In addition to the scientists covered in *Natural Computing*, the following scientists will also be covered:

- Alan Turing
- Charles Babbage / Ada Lovelace
- John von Neumann

Next week, we will be signing up for which scientists we would like to present.

Assignments

- Read 1st Chapter in *Natural Computing* discussing Rodney Brooks
- Pick a scientist you would like to present from the book

Mathematical Terminology

∀	for all
{ n l n is even }	the set of n such that n is even
∃	there exists at least one
∈	is a member of

Gottlob Frege

A German mathematician named Gottlob Frege wrote *Grundgesetze der Arithmetik (The Ground Truth of Arithmetic)*. The purpose of the work was to form foundational laws for all of arithmetic that were consistent and constructible. Bertrand Russell wrote to Gottlob shortly before the publication of his book. Russell, then 19 years old, pointed out in his letter that the axioms described by Gottlob led to contradictions.

Bertrand Russell

As a response to Gottlob's book, Bertrand Russell introduced the Russell Set:

```
{ x | all sets that do not have themselves as a member }
```

Consider the following two sets:

1) A set of teacups that holds only teacups

```
{Joe's teacup, Sally's teacup, ...}
```

Because the set of teacups itself is not a teacup, it does not belong in the set of teacups.

2) A set of non-teacups that holds anything but teacups

```
{chair, airplane, {non-teacups}, ...}
```

Because the set of non-teacups itself is not a teacup, it belongs in the set of non-teacups.

The Russell Set is used to demonstrate the following question: is RS $\in$ RS? The question produces a paradox. If RS is indeed a member of RS, then it does not satisfy it's own condition. However, if RS is not a member, then it should be a member to satisfy the condition. Thus, the question highlights the contradiction present in Gottlob's *Grundgesetze der Arithmetik*. Gottlob's book was still published but with an appendix acknowledging the paradox found by Russell and the lack of a solution.

Principa Mathematica

Bertrand Russell and Alfred Whitehead (both Englishman), wrote a book called *Principa Mathematica*. In it they suggested a solution to solve Gottlob's problem: disallow self-referential comprehension. The book did not attract many readers but fortunately was read by Kurt Göddel. Göddel came to the conclusion that a system powerful enough to describe all of mathematics was impossible.

`Note: Principa is pronounced with a hard c`

Alan Turing

Alan Turing, a mathematician from Princeton University, wanted to build a computer. According to Alan Turing, a computer can be defined as a person, the computer, who sits between two piles of paper. The person can choose to retrieve a piece of paper from either pile and read and write on it. The person can then place the paper onto either pile.

From this definition, Alan Turing derived the Turing Machine:

The machine consists of a tape head and a roll of tape. The tape head can read or write on the current tape, or move left or right to a different piece of tape. The tape holds data or instructions for programs.

Properties of Computers

- Computers have finite instructions
- Anything possible with a modern computer can be done with a Turing Machine

`Alan turing later went on to decrypt intercepted German Navy messages during WWII.`

Entscheidungsproblem

A German mathematician named David Hilbert wanted an arithmetic, logic-based foundation for mathematics. He posed the Entscheidungsproblem (translated to the *decision problem* in German), which questions whether an algorithm exists that can prove or disprove any specific mathematical assertion.

Alan Turing asked a similar question: What if there's a program that could ask questions about programs?

The Halting Problem

Knowing when programs will stop is important in mission critical software used in rockets, airplanes, power plants, and other safety dependent environments. However, is there a way to know whether any program has stopped executing or is still computing? Simply put, no.

Consider a program T which is defined as "translates lowercase letters to uppercase". For example `T(word)` will produce "WORD". T can also be run on itself, as `T(T)`, which produces "TRANSLATES LOWERCASE LETTERS TO UPPERCASE." The fact that T can run with the input T shows that programs can be self-referential.

Another program exists called Halt, which takes in a program and the input of that program. The Halt program produces yes or no depending on whether or not the program will stop. When `Halt(T, input)` is run, the program produces a yes. This is because T will always stop provided that the input is finite.

A derivative of Halt is HaltCrazy. Turing states that if Halt is definable, the same is true for HaltCrazy. The program can be represented in the following pseudocode:

HaltCrazy(program)

```
if Halt(program, program) is yes
    run forever
else
    stop
```

Although Halt and HaltCrazy are shown to be definable, consider what happens when `Halt(HaltCrazy, HaltCrazy)` is executed. If `HaltCrazy(HaltCrazy)` stops, then `Halt(HaltCrazy, HaltCrazy)` returns yes and thus `HaltCrazy(HaltCrazy)` should go on forever. However, if `Halt(HaltCrazy, HaltCrazy)` returns no, `HaltCrazy(HaltCrazy)` should stop.

This poses a contradiction which shows that Halt, similar to the Russell Set, is not well-defined. From this, we can conclude that **The Halting Problem** cannot be computable.