Math is Turing Complete

Ethan Holland Simmons

June 26, 2020

*Abstract*

*With computers everywhere, creating them can seem to be very hard. The engineering and amount of research to make high quality cpu’s is overwhelming. While this is nowhere close to the answer, it demonstrates that computations can happen in the least likely places and on the least likely pieces of hardware.*

## 1.) Introduction:

This concept builds on 2 past ideas ([nested numbers](https://sites.google.com/view/ethan-s/nested-numbers), [boolean reductions](https://sites.google.com/view/ethan-s/boolean-reductions)), so I recommend looking at those.

I’ve always loved looking for random things that are [accidentally Turing complete](https://beza1e1.tuxen.de/articles/accidentally_turing_complete.html). With the complexity of math and all of the many, many structures, there must be a way to make this into a computer. I had readable/writable memory and boolean expressions. The only thing left was looping to make this into a computer. After many failed attempts (recursive piecewise functions and summations to name a few) I used an idea from *Mov is Turing Complete* by Stephan Dolan. With infinite memory, we can infinitely repeat execution.

## 2.) The Pseudo Turing Machine:

### Notation:

The square brackets indicate reading an index from a list (starting at index 0). This list and index idea used here is explained in the nested numbers paper. The curly brackets indicate creation of a new list (as seen in java). The line means creating a list that has 0 at index 0, and each following index corresponds with of (index 1 is index 0).

## 3.) Turing Machine

## 4.) Explanation:

### Restrictions:

In mathematical notation, variables are static. Once declared, they can’t change value again. They are not ‘variable’ but that is the common name in programming, so that’s what I’ll call them. This causes an issue with a writable memory. With infinite space though, we can create copies of variables with the proper edits. We also don’t have access to any looping structures at all, so we have to repeat the executed instructions. Conditional branching is also not included normally, but with boolean reductions, the conditional write/move/goto can be achieved. is always binary, so multiplying by the values to be used if is 1, and multiplying negated () to the values to be used if 0 will evaluate to the proper values. Memory is a binary list (nested number) of an infinite size. Lower indices need to increase to accommodate for the moving read/write head, while the upper indices are infinite and 0.

### Writing Instructions:

The list is a list of all of the instructions and is the constant used to read (see nested numbers). The instructions follow the format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| write | move | goto | write | move | goto |
| If 0 | If 0 | If 0 | If 1 | If 1 | If 1 |

The pseudo turing machine:

Can be seen as:

The domain for each value is as follows:

(add the value to the memory cell. It’s ternary to write a 0 to a 1 by using -1)

(positive for right, negative for left. This is not really important)

(go to any instruction by number. 0 is halt)

(the memory is binary as in a normal turing machine)

(the constant needs to be larger than any other number in the instructions)

### Example instructions:

This is an example instruction set to write 1 to the cell if it sees a 0 and go positive. If it sees a 1, it doesn’t change the cell, goes negative and it halts.

### Halting:

The machine will never halt, but rather reach a halting value of . When , the person running the machine can know the machine is done. To get the last value of the ‘tape’, one needs to evaluate () or when .

### Running:

The first block is initialization. It gives instructions, starts memory at all 0’s, starts on instruction 1, and points to memory cell 0. The variable is to contain the memory cell pointed at by . The next block is the first execution round. Memory is written to, the memory pointer moves, and the instruction pointer is updated. The memory is expanded because the memory tape needs to be infinite. The memory pointer can only increase or decrease by 1 each round, so a lower index needs to be added (there are infinite upper indexes with nested numbers). The memory pointer then needs to be moved up one to accommodate for the changed memory. Lastly, the next current memory cell is stored. The program is an infinite copy of the execution round preceded by an initialization block. Each round follows the same format, increasing subscripts and summation max.

## 5.) Conclusion:

Math is turing complete. Not just any math, but only up to high school math (or calculus for summations and modulus). The only concepts that are used are:

* Variables ()
* Modulus division
* Exponentials
* Division
* Floor (or rounding down)
* Addition
* Multiplication
* Summation
* Subtraction

This allows any device with the ability to do math (calculator hint hint) to act as a computer and compute problems way above its designed purposes.

TODO: find a way to change a single index of a at each stage (overwrite 1 cell of memory; check during summation? YES keep the a` variable and m to insert the value once n=m; check at every step of sum)