Linearity (MATH 526) Final Project

Max Miller[1]

[1]University of New Hampshire

May 15, 2018

Abstract

Alan Turing's paper "On Computable Numbers" (**?**) provides a framework for Computing Machines, which are physical symbol manipulation machines, which can, in theory, compute a specific computable sequence if programmed to do so. Each of these machines, subsequently named "Turing Machines" is provided with a read-write head, a programmable structure, and input tape of symbols, which the machine can move left or right. From this concept, Turing asserts "It is possible to create a single computing machine which can be used to compute any computable sequence." referring to what he calls a Universal Computing Machine. The purpose of my project is to explore this theory from first principles and create a testing framework to accurately display the computational ability of Turing Machines. I broke up the learning into two sections: computational theory and programming language/compiler design.

PyTuring

For my work on Turing's assertation in On Computable Numbers, I thought it appropriate to create a programming language which would virtually implement the physical architecture of his proposed computing machine. PyTuring is the first iteration of this programming language, created in late March. The language consists of a four-step command structure with read, write, move, and state changing capabilities. At conception, the language allowed for a finite, discrete tape with a Unicode-8 alphabet. Command syntax started (and stayed) as follows: {STATE} {TAPE VALUE} {ACTION} {NEW STATE}. This structure is based on Turing's own structure for the command states of his computing machines. For example, you would read the line:

```
0 A >> 1;
```

As: If in the current machine is in state 0, and the tape value at the current read-write head position is A, move the read-write head right, and move into state 1. The semicolon denotes the end of a command. With such a structure, it is apparent how a larger program could be created with the correct input tapes and commands to compute larger systems. By nature, this language is Turing complete.

Of course, any good computer program is not only a series of commands but also allows for more "quality of life" features for the programmer. PyTuring allows for submachines, denoted by SUB and ENDSUB commands (see documentation). In Turing's paper, his notes the Universal Computing Machine as a machine which can emulate any computing (Turing) machine, and submachines in PyTuring are a way to represent this concept. There is no real benefit to using a submachine, except that they provide a great deal of help in code organization, allowing for internal state keeping and dynamic naming. A submachine can be declared and called as such:

```
SUB add:
    ...
```

```
ENDSUB
0 call add 1;
```

Compiling PyTuring

The larger opportunity of testing Turing's assertion is learning the structure and methods of programming languages, and how many of the ones I use every day operate. As an additional task to PyTuring, I also sought to build a compiler to convert the language into x86 Assembly. This task proved to be even more challenging than creating the language, as it required me to begin learning x86 Assembly, as well as proper compiler design and structure. I used Compilers: Principles, Techniques, and Tools, referred to as "The Dragon Book" as my primary tool for learning about compilers. Although I did not create a complete, efficient, or even well-working compiler for PyTuring, the book taught me quite a lot in regards to how a programming language can be structured, effective design techniques, and the purpose of compiled languages. As a result, the book greatly influenced my design of PyTuring, especially in regards to preparing the language to be a compiled one.