

Thesis Proposal

Adam Yedidia

April 20, 2015

Abstract

For my thesis, I propose to write a general-purpose compiler from a programming language of my design to a single-tape, two-symbol Turing machine with as few states as possible. Once this is complete, I propose to write a program in my language that will verify the consistency of Zermelo-Fraenkel set theory (or ZFC), and compile that program down into a Turing machine description. In so doing, I will have created a description for a Turing machine whose behavior cannot be predicted by ZFC. This is true by inspection, because since a logic system cannot prove its own consistency, a logic system cannot prove whether or not a Turing machine that verifies its consistency accepts or rejects.

0.1 Introduction

The work of my thesis would not have been possible without first splitting it into two parts: the construction of the compiler, and the

Chapter 1

The Compiler

The compiler used and referenced in this paper has as its goal to convert a program written in TMD into a description of Turing machine running on a single tape and using a 2-symbol alphabet. In between, the code is passed through two intermediate stages: a description of a Turing machine running on multiple tapes using a 3-symbol alphabet, and a description of a Turing machine running on a single tape using a 4-symbol alphabet. A detailed diagram illustrating this process is visible in Figure 1.1.

In this chapter, each of the four representations will be described in turn, from highest- to lowest-level. Additionally, detailed explanations will be given for how the conversion of one representation to the one immediately below it is done.

1.1 TMD

The top-level representation is a program written in the TMD language, which is a language created and designed explicitly for use in this project. TMD is altogether not unlike Assembly Language, although it is more powerful in some ways and weaker in others.

There are two types of TMD files. The first type of TMD file is a TMD *main file*. A main file is a file that can be run on its own, but cannot be called by another program. The second type of TMD file is a *function file*. A function file can be called by another program, but cannot be run on its own. The two types of files obey differing syntax, and the differences between them will be explained in further detail in this section.

A TMD program is a sequence of commands. Commands are separated by newlines. Each command is given its own line.

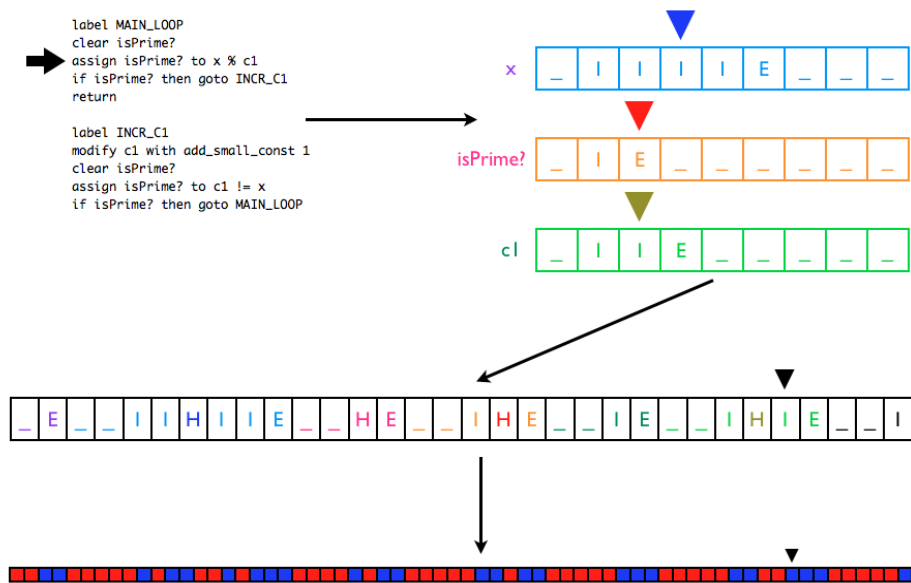


Figure 1.1: This diagram illustrates each of the steps in the conversion between a program written in TMD and a description of a single-tape Turing machine with a binary tape alphabet.

1.1.1 List of Commands

The following is a list of possible commands. Let **x**, **x1**, **x2**, and **x3** be the names of distinct variables, let **c** be a numerical constant, and let **f** be the name of a function. Let **L** be the name of a code label.

Main files only

var x: Declares a variable with the name **x**.

vars x1 x2 ...: Declares several variables with the names **x1**, **x2**, ...

Function files only

input x1 x2 ...: Defines the arguments to the function file to be **x1**, **x2**, ...

return: Exits the function and returns to executing wherever the function was called.

Either file

assign x1 to x2 [operation] x3: Changes the value of the variable **x1** to the result of **x2 [operation] x3**.

assign x1 to x2 [operation]: Changes the value of the variable **x1** to the result of the operation **[operation]** applied to **x2**.

modify x1 with [operation] x2: Changes the value of the variable **x1** to the result of **x1 [operation] x2**.

modify x with [operation] c: Changes the value of the variable **x** to the result of **x [operation] c**.

clear x: Changes the value of the variable **x** to 0.

label L: Declares a label **L**.

goto L: Changes the line of code being executed to the line of code that contains “**label L**.”

if x goto L: If the current value of **x** is a positive integer, then changes the line of code being executed to the line of code that contains “**label L**.”

function f x1 x2 ...: Calls the function **f** with the arguments **x1**, **x2**, ...

accept: The Turing machine accepts.

reject: The Turing machine rejects.

1.1.2 Main Files

Main files contain the body of a program, and can be run. Running a main file will cause the sequence of commands in that main file to be run, starting at the top of the main file.

Any variable declared at any point in a main file will be initialized before the execution of the program, with an initial value of 0. As part of their execution, main files can call function files, but they cannot call other main files.

1.1.3 Function Files

Function files contain the description of a function.

```

vars even_number c1 h1 h2 h3 isPrime?
modify even_number with add_small_const 4

label EVEN_NUMBER_LOOP
  clear c1
  modify c1 with add_small_const 2

  label C1_LOOP
    clear h3
    label LOAD_C1
    clear h1
    assign h1 to c1
    goto RUN_ISPRIME
    label LOAD_C2
    modify h3 with add_small_const 1
    clear h1
    assign h1 to even_number
    modify h1 with - c1
    clear h2
    assign h2 to h1 equals_small_const 1
    if h2 then goto REJECT
    goto RUN_ISPRIME
    label RUN_ISPRIME
    function isprime h1 h2 isPrime?
    if isPrime? then goto CHECK_H3
    goto INCR_C1

    label CHECK_H3
    if h3 then goto INCR_EVEN_NUMBER
    goto LOAD_C2

    label INCR_C1
    modify c1 with add_small_const 1
    goto C1_LOOP

    label INCR_EVEN_NUMBER
    modify even_number with add_small_const 2
    goto EVEN_NUMBER_LOOP

label REJECT
reject

```

Figure 1.2: This is the TMD code for a parsimonious program that will loop if Goldbach's conjecture is true, and reject if it is false. It is provided as an example of a TMD main file. See Figure 1.3 for the code of the `isprime` function, which is called by the main body of the program.


```

input x c1 isPrime?

label this Function can't handle having 1 get passed in
label isPrime? is the boolean that gets set to whether or not x is prime

clear c1
modify c1 with add_small_const 1

goto INCR_C1

label MAIN_LOOP
clear isPrime?
assign isPrime? to x % c1
if isPrime? then goto INCR_C1
return

label INCR_C1
modify c1 with add_small_const 1
clear isPrime?
assign isPrime? to c1 != x
if isPrime? then goto MAIN_LOOP
modify isPrime? with add_small_const 1
return

```

Figure 1.3: This is the TMD code for a function that will set the value of the input `isPrime?` to 0 if the input `x` is not a prime number, and will set the value of `isPrime?` to a positive integer if `x` is a prime number. The value of `x` is not modified by the `isprime` function, but the value of variable `c1` may be changed to something arbitrary and the value of `isPrime?` will be changed to depend on the primality of `x`. The function's result will depend exclusively on the initial value of `x`.