TMD Documentation

Adam Yedidia

March 10, 2016

The top-level representation is a program written in the TMD language, which is a language designed to give the user a simple interface with which to program a multi-tape, 3-symbol Turing Machine with a function stack.

TMD code can be processed in two ways. First, it can be *interpreted*; that is, it can be directly evaluated line-by-line. This is generally done to verify a program's correct behavior, and to correct errors which would result in thrown exceptions in the interpreter but might lead to undefined behavior in the compiled Turing machine (because the compiled Turing machine is optimized for parsimony, whereas the interpreter need not be).

Second, TMD code can be *compiled* down to a description of a single-tape, 2-symbol Turing machine. It is highly recommended, however, to first interpret any piece of TMD code before compiling it, because the interpreter is much better for catching programming errors. The compiler is general-purpose and not restricted to compiling the programs discussed in this thesis. It is optimized to minimize the number of states in the resulting Turing machine, not to make the resulting Turing machine time- or space-efficient.

A TMD program is contained in a directory. It is a collection of *functions*, each of which contains a sequence of *commands*. Each function is given its own file. Commands are separated by newlines. Each command is given its own line.

A TMD program also contains two special files which contain important information about the multi-tape Turing Machine: a file named `functions.tff` and a file named `initvar`.

What follows is a brief description of the syntax of the TMD language. A much more detailed description is available at [?]. The brief description contained here should be enough for a reader to gain a reasonable understanding of the programs written for this paper; any reader wishing to write her own programs is strongly encouraged read the more detailed documentation before proceeding.

1 Function Files

Most of the logic contained within a TMD program lies in its function files. TMD functions can make reference to the tapes in the machine and read the symbol under the head of each tape, much as a standard multi-tape TM can. In addition, TMD functions can call other TMD functions, pushing those functions to the top of the stack. Finally, TMD functions can return, popping themselves from the function stack.

A TMD function file is composed of four kinds of lines: an input line, calls to other functions, tape commands, and return statements.

1.1 The Input Line

Every TMD function file begins with an input line. TMD function files cannot contain more than one input line. The input line describes what tapes are going to be passed in as arguments to the function, and will therefore be readable and writeable within the function. An example of an input line might be:

```
input x y z
```

1.2 Labels

Any line in a TMD function other than the input line may be preceded by a *label*. A label takes the following form:

[alphanumeric string]: [line of code]

A label is an indicator attached to a line of code. Using tape commands, the line of code can be referenced in order to jump to the labelled line of code. The label can also be given a descriptive name that helps to explain the purpose of the line. This is perfectly legitimate even if the programmer has no intention of jumping to the line later; the addition of a label results in no additional states in the compiled program.

1.3 Function Calls

TMD functions can contain calls to other functions. Function calls contain the name of the function being called, followed by which tapes will be passed as arguments to the function. An example of a function call might be:

```
function add x y
```

1.4 Tape Commands

Tape commands begin with a marker that indicates which tape is going to be used in the command. They then explain what to do if each possible symbol is read.

In TMD, there are three legal tape symbols; they are `1`, `E`, and `_`. `_` is the empty symbol (the only one which can appear an infinite number of times on any tape). Furthermore, in TMD, it is a requirement that each tape always have the form $(\_)^{\infty}(\texttt{1}|\texttt{E})^{+}(\_)^{\infty}$ (here, the $(.)^{+}$ operation denotes repetition any positive number of times, and the $(.)^{\infty}$ operation denotes repetition an infinite number of times). This requirement is for ease of compilation down to a single-tape TM; later, when the TMD is compiled down a Turing machine and that Turing machine is searching the registers for a register that matches a specific ID, the search will assume that the register organization will alternate (identifier, value, identifier, value) and that a value will be represented as a "chunk" of 1's and E's. This is why the above requirement must be respected.

Tape commands have the following form:

`[`[variable name]`]` ([symbol read] `(` [list of reactions] `)`)*

The list of reactions is a series of words separated by commas. These reactions can include what direction to move, what symbol to write, and what line to jump to after execution.