Turing Machine Simulation Language

Language Reference Manual

Isaac McGarvey (iam2108)
Joshua Gordon (jbg2109)
Keerti Joshi (kj2217)
Snehit Prabhu (sap2131)

Table of Contents

1 Introduction
2 Lexical Conventions
2.1 Tokens
2.2 Comments
2.3 Symbols
2.4 Symbol Lists
2.5 Keywords
2.6 Separators
2.7 Data Types
2.8 Variables
3 Declarations
4 Statements
4.1 If Statements
4.2 While Statements
4.3 Write Statements
4.4 Left Statements
4.5 Right Statements
4.6 Exit Statements
4.7 Statement Lists
4.8 Program Structure
5 Scope
6 Functions
7 Sample Program
7.1 Sample Code
7.2 Transition Diagram
7.3 Equivalent Interpreted Code

1. Introduction

A Turing Machine is a simple but powerful computer. It is useful in thinking about the nature and limits of computability because its method of computation is about as simple as can be imagined. Important theoretical results about what can be computed that are expressed in the

terms of Turing Machines are clearer to intuition than the same results expressed in other terms.

The Turing Machine is an automaton whose temporary storage is a tape. This tape is divided into cells each of which is capable of holding one symbol. Associated with the tape is a read-write head that can travel left of write on the tape and that can read or write a single symbol on each move. The automaton that we use as a Turing Machine will have neither an input file nor any special output mechanism. Whatever input and output is necessary will be done on the machine's tape.

The Turing Machine Simulation Language (TMSL) is a programming language that is designed allow users to write programs that will be compiled into a Turing Machine that can execute the program. Because Turing Machines are very different from normal computers and also far more restrictive in terms of the number of available operations, TMSL is very different from a typical imperative programming language. Programming a Turing Machine by specifying states and transitions is analogous to programming a modern computer in assembly, at best. We hope that TMSL will be a high-level language (relatively speaking) for Turing Machines.

2 Lexical Conventions

2.1 Tokens

There are three types of tokens in this language: identifiers (symbols), keywords,and separators. Tokens are whitespace delineated.

2.2 Comments

TMSL supports only single line comments. A line beginning with an @ symbol is a comment. An example comment is as follows:

@ this is a comment
@ this is another comment

2.3 Symbols

Symbols are values / identifiers that can be read from and written to the tape.
A symbol is represented by a sequence of letters and digits. Letters are not case sensitive.

<symbol> = ['a'-'z' 'A'-'Z' '0'-'9']+

The only only special symbol is the symbol / keyword *'_' (to represent a blank*). All uninitialized cells of the tape contain the blank symbol. As with any other symbol, the '_' symbol can also be read from or written to the tape.

The set of all symbols a machine can read and write, excluding the '_' symbol, are called its alphabet.

2.4 Symbol Lists

A symbol list is a sequence of symbols separated by commas:

<comma> = ','
<symbol list> = (<symbol> <comma>)* <symbol>

2.5 Keywords

The following are reserved for use as keywords and may not be used as identifiers:

if
while
left
right
write
exit
'_'

2.6 Separators

The following are used to indicate separators:
,
{ }

2.7 Datatypes

The only data type in TMSL is that of the set of symbols that the machine can process. Each of the implicit variables in TMSL (that is, the cells on the tape) are of this data type by definition.

2.8 Variables

TMSL does not have any explicit variables. The cells of the tape act as variables, but only the cell that is currently underneath the head can be accessed. All reading and writing of the tape cells implicitly access the current cell of the tape, as in the condition part of the control statements and in the write instruction.

3 Declarations

The user implicitly defines the symbols which are allowable in the alphabet via statements (e.g., if a statement references a symbol, then that symbol is considered part of the alphabet). The empty symbol ('_') is considered a member of the alphabet by default.

4 Statements

A statement in TMSL can be one of the following types: if, while, write, left, right,exit

<statement> = <if-statement> | <while-statement> | <write-statement> | <left-statement> | <right-statement> | <exit-statement>

4.1 If

An if statement in TMSL looks like this:

if (<*symbol list*>)
{
<*statement list*>
}

This means that if the symbol in the current cell of the tape is contained in the symbol list, the statement list is executed. Otherwise, the statement list is not executed. In either case, execution then continues with the next statement.

<if-statement> = 'if' '(' <symbol list> ')' '{' <statement list> '}'

4.2 While

A while statement looks like this:
while (<*symbol list*>)
{
<*statement list*>
}

This means that the statement list is executed repeatedly, as long as the symbol at the current cell is in the symbol list.

<while-statement> = 'while' '(' <symbol list> ')' '{' <statement list> '}'

4.3 Write

A write statement looks like this:
write <*symbol*>

This writes the the symbol onto the current cell of the tape.

<write-statement> = 'write' <symbol>

4.4 Left

A left statement is simply the left keyword itself:
left

This statement moves the tape one cell to the left.

<left-statement> = 'left'

4.5 Right

A right statement is simply the right keyword itself:
right

This statement moves the tape one cell to the right.

<right-statement> = 'right'

4.6 Exit

An exit statement is simply the exit keyword itself:
exit

This statement halts the program execution immediately.

4.7 Statement Lists

A statement list is simply a sequence of statements. There is no separator necessary between the statements.

<statement list> = <statement>+

Semantically, each statement in a statement list is executed in order.

4.8 Overall Program Structure

A program in TMSL is simply a list of statements, executed in order.

5 Scope

Since there are no explicit variables in TMSL, there are no scope rules. The only thing that might be considered to have a scope are the symbols in the machine's alphabet. The scope of the alphabet symbols are global.

6 Functions

There are no built-in or user-specified functions in TMSL.

7 Sample program

<u>Sample program for computing proper subtraction</u>

Proper subtraction over natural numbers rounds off any negative integer outcome to 0. (i.e., m - n = 0 for all m>n). The following illustrates an imperative program for proper subtraction in our language, which is then interpreted into TM transitions. For a similar example, refer to page 325 of Ullman, Motwani

The Turing machine has 1 tape, on which we provide an input, and on which it solely operates till it halts.

Tape Alphabet of the TM is {0,1}, with the additional default blank space character '_'.

Movements of the tape head are {L,R,N}, representing left, right and no-op (stay in current position)

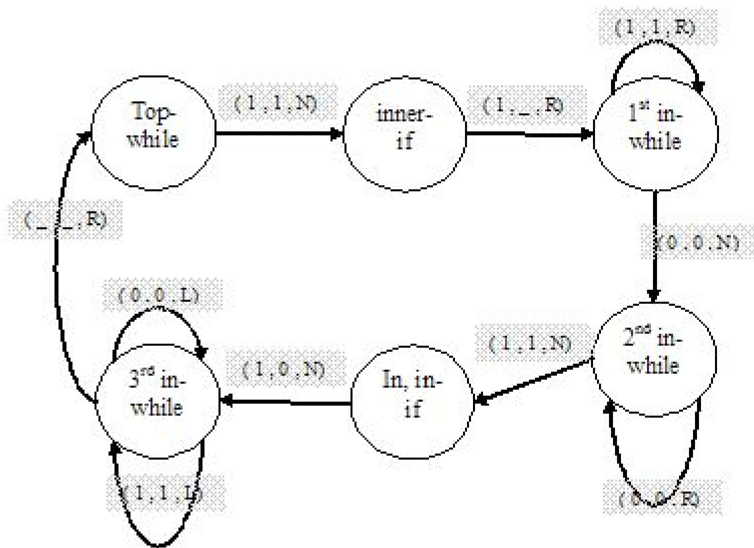
Input to the program is of the form 1^m01^n , when the user wants to perform (m – n).

Output is what remains on the tape at when the Machine halts.

7.1 Sample code in our language:

```
@while there are '1' characters remaining in the first number.
while (1) {

    if(1) {
        @ mark this character as seen.
            write _

        @now move right till the next number (past the intervening 0s)
        while (1)  {
                right
            }
            while (0) {
                right
            }

            @strike off one character of this number as well.
            if (1)  {
                write 0
            }

            @return to the first number
            while (0 or 1) {
                left
            }

    @go left all the way till you encounter a _
    right
    }

}
```

7.2 Transition Diagram:



7.3 Equivalent Interpreted Code

The Transition Table for the above diagram captures the code. In the absence of an appropriate transition, the machine halts.

Sample Input : 1111011_ _ _ …
Output : _ _ 11000 _ _ _ …