Turing Machine implementation in Processing instruction set documentation
Or:
Overkill: the lab project

Fionan Haralddottir

November 2, 2014

Overveiw

This is the documentation written for the instruction set designed for an implementation of a Turing Machine written as a personal project during October 2014.

I started the project because I felt that I had understood enough information about how Turing Machines worked that I could implement one, I chose to write my implementation in Processing as it would be easy to give a visual representation of the machine's progress.

I do understand how elitist I'm behaving writing a first-year project doc in latex instead of a webpage or text file but I'm bored, so this is how I'll be.

It is also entirely possible that I've misunderstood some fundamental parts of how a turing machine operates, making this documentation and the program itself very pretentious.

1 Documentation

1.1 Formatting

The instruction set is case insensitive, except for "[PAGE N]" and "[END-PAGE]" which both need to be in uppercase.
Newlines are the terminating character for an instruction, it doesn't matter how many other whitespace characters seperate one argument from another

or if the line starts with a series of tabs, that instruction should still be interpreted.

1.2 Pages

```
[PAGE 0]
... // instructions
[ENDPAGE]

[PAGE 1]
... // instructions
[ENDPAGE]
```

In this implementation a program is organised into a set of pages, each page holds a set of statements in the order they will be executed. Each page is numbered, with 0 as the lowest possible page number and 999 as the highest possible.

"PAGE 0" is the page that each program must include, it is the page that is called at the beginning of the program.

A page declaration is notated by "[PAGE N]" where N is the page number, followed by a set of commands with a terminating "[ENDPAGE]".

1.3 Comments

Comments in a program are shown by a double forward-slash, like in C/++ and Java, but without multi-line comments. anything between these and a newline are ignored by the interpreter.

```
[PAGE 0]
//this is a comment, and will be ignored
flip
call 0
[ENDPAGE]
```

1.4 SCAN

```
...
scan 3 //move 3 bits right
...
scan -8 //move 8 bits left
```

...

The "SCAN" instruction moves the head of the Turing Machine the specified number of bits along the tape, this is bi-directional so it can be either positive or negative integers for moving back and forth. The argument can be any positive or negative integer, but it will take a number of steps equal to the absolute of that value to get to that position.

1.5 CALL

```
[PAGE 0]
flip
ifbit
        call 1 //page 1 is called
endif
[ENDPAGE]

[PAGE 1]
call 0 //page 0 is called
[ENDPAGE]
```

The "CALL" instruction begins the execution of another (or the same, recursion!) page in the program, once that page has finished executing then the page that originally called it will continue running.

1.6 FLIP

```
[PAGE 0]
flip //inverts the value of the current bit
[ENDPAGE]
```

The "FLIP" instruction inverts the value of the bit at the head of the Turing Machine, if the bit is 1 it becomes 0, if 0 then 1.

1.7 IFBIT and ENDIF

```
[PAGE 0]
//example if statement
ifbit
        //set of statements to execute
endif
//example else, or if-not statement
```

```
flip
ifbit
        //set of statements to execute
endif
flip //might not be original bit
[ENDPAGE]
```

The "IFBIT" and "ENDIF" instructions are the control flow statements for this instruction set. The instructions in the body of the IFBIT statement will only be executed if the bit the head of the Turing Machine is 1, otherwise all instructions are skipped until the next matching "ENDIF". These statements can be nested.

1.8 HALT

```
[PAGE 0]
ifbit
        halt
endif
... //do other things
[ENDPAGE]
```

The "HALT" instruction halts the machine before it can reach the end of the program.

1.9 SAVE and SPOP

```
[PAGE 0]
flip
save
lgic xor #0
spop
[ENDPAGE]
```

The "SAVE" and "SPOP" instructions are the instructions related to the Turing Machine's registry of up to 32 bits. "SAVE" will add the value at the head of the machine to the top of the stack, while "SPOP" will remove the item at the top of the stack. These values can be referenced during a "LGIC" instruction by a relative index, "#0" will reference the item on the top of the stack, "#1" will reference the item one below the previous, and so on.

1.10 LGIC

```
[PAGE 0]
```

```
flip
save
scan 1
lgic xor #0
```
[ENDPAGE]

The "LGIC" instruction is the instruction that deals with logical operations and is the only instruction that takes two arguments. There are 4 logical operators availiable to use: "AND", "OR", "XOR", and "NOT". Besides "NOT", which just sets the bit as the inverted value of the stack item, each operator applys a logical operation betwen the bit at the head of the machine and bit referenced in the registry.