# Introduction to Esoteric Language Malbolge

Masahiko Sakai (Nagoya University)

10 Dec 2010, College of Technology, VNU



## 99 bottles of beer(1/3)

- http://www.99-bottles-of-beer.net/
- Collection of programs that output the song

```
99 bottles of beer on the wall, 99 bottles of beer.
Take on down and pass it around, 98 bottles of beer on the wall.

...

2 bottles of beer on the wall, 2 bottles of beer.
Take on down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take on down and pass it around, no more bottles of beer on the
wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.
```

- Programs in more than 1200 different languages



## 99 bottles of beer(2/3)

- Top list (2008): ours in Malbolge is No.1 !

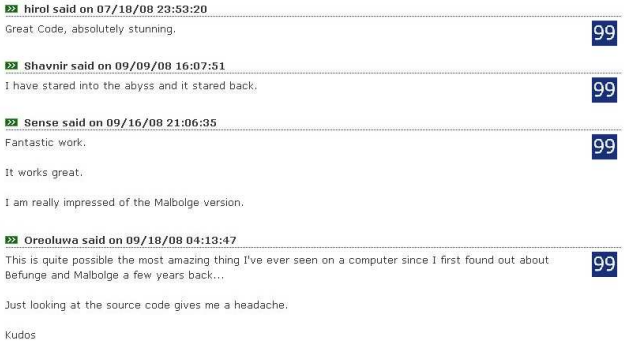

**Top Rated**

| # | Language | Author | Date | Comments | Rate |
|---|---|---|---|---|---|
| 1. | Malbolge (real loop version) | Hisashi Iizawa | 12/29/05 | 32 | ★★★★★ |
| 2. | Piet | Dana Connell | 09/10/06 | 19 | ★★★★☆ |
| 3. | LOLCODE (sjlol version 1.1 for lolcode 1.0) | Mike Gogulski | 07/13/07 | 15 | ★★★★☆ |
| 4. | Whitespace | Andrew Kemp | 04/20/05 | 15 | ★★★★☆ |
| 5. | Haskell (Using guards) | Simon Johansson | 10/25/07 | 1 | ★★★★☆ |
| 6. | Shakespeare | Jonas Sjöbergh | 05/18/05 | 6 | ★★★★☆ |



## 99 bottles of beer(3/3)

- Comments for Malbolge






## Esoteric Programming Languages

- Language designed to be HARD to read and write
  - INTERCAL [Woods, Lyon 1972]
  - BrainF∗∗∗ [Müller 1993]
  - Befunge [Pressey 1997]
  - Malbolge [Olmstead 1998]
- Proposed as jokes
- References

  http://members.tripod.com/rkusnery/weird.html

  http://www.99-bottles-of-beer.net/toplist_esoteric.html



## Malbolge

- A programming language come from HELL
  - 10 trits machine defined by its interpreter
  - Each operation is modified after execution
  - Only one trit-wise operation is poor

    A,[D] := $crz$(A,[D])

| [D] \ A | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 2 |
| 2 | 2 | 2 | 1 |

  - Hard to load data, because non-operations are not loadable.

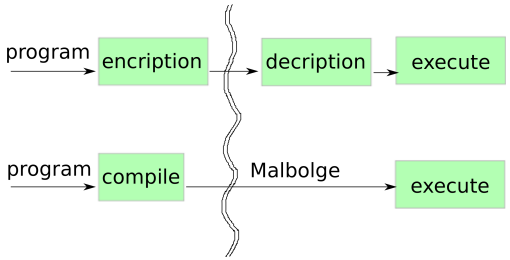
## History of Malbolge

- [Ben Olmsted 1998] Language Proposal
- [Andrew Cooke] "HEllo WORld" program
- [Anthony Youhas 2000] Three programs that output strings
- [Lou Scheffer] Program like "cat"
- [Tomasz Wegrzanowski 2004] Method to produce a program that outputs a given string



## Our Motivation

- Programs written in Esoteric Languages are like encrypted but Executable without encryption






## This talk

- Explanation of Malbolge
- Overview of Programming technique
  - Bootstrapping
  - Low level assembly language
  - High level assembly language




## Syntax of Malbolge

- String of printable characters (33 to 126 in ASCII) where spaces are ignored.
- Each i-th character x must be an operator

```
const char xlat1[] =
  "+b(29e*j1VMEKLyC})8&m#~W>qxdRp0wkrUo"
  "[D7,XTcA\"lI.v%{gJh4G\\-=O@5`_3i<?Z'"
  ";FNQuY]szf$!BS/|t:Pn6^Ha";

if(strchr( "ji*p</vo",
    xlat1[( x-33+i ) % 94]) == NULL)
  fputs("invalid character\n", stderr );
```

Note that illegal operator is not loadable, but executable as Nop.



## Example of Malbolge program

- "HEllo WORld" program by Cooke.

```
% cat hello-fake.mal
(=<`$9]7<5YXz7wT.3,+O/o'K%$H"'~D|#z@b=`{^L
x8%$Xmrkpohm-kNi;gsedcba`_^]\[ZYXWVUTSRQPO
NMLKJIHGFEDCBA@?>=<;:9876543s+O<oLm
% ./malbolge hello-fake.mal
HEllO WORld
```

- Operations obtained by xlat1[(x-33+i)%94]

```
jpp<jp<pop<<jo**popp<o*p<pp<pop<pop<jijoj/
o<vvjpopoopo<ojo/ovooooooooooooooooooooooo
ooooooooooooooooooooooooooo*p<v*<*
```



## Semantics of Malbolge(1/3)

- Malbolge machine
  - Address area = One word = ten trit
    0 to $59048 = 3^{10} - 1$
  - Codes and data are stored in memory
  - Three registers
    - A: accumulator
    - C: code pointer
    - D: data pointer

## Semantics of Malbolge(2/3)

- Execution

```
const char xlat2[] =
 "5z]&gqtyfr$(we4{WP)H-Zn,[%\\3dL+Q;>U!pJS72FhOA1C"
 "B6v^=I_0/8|jsb9m<.TVac`uY*MK'X^xDl}REokN:#?G\"i@";
for (;;) {
  if ( mem[c] < 33 || mem[c] > 126 ) continue;
  switch ( xlat1[( mem[c]-33+c )%94] ) {
    case 'j': d = mem[d]; break;
    case 'i': c = mem[d]; break;
    case '*': a = mem[d] =
                mem[d]/3 + mem[d]%3*19683; break;
    case 'p': a = mem[d] = crz( a, mem[d] ); break;
    case '<': putc( a, stdout ); break;
    case '/': x = getc( stdin ); a = x; break;
    case 'v': return;
  }
  mem[c] = xlat2[mem[c] - 33];
  if ( c == 59048 ) c = 0; else c++;
  if ( d == 59048 ) d = 0; else d++;
}
```



## Semantics of Malbolge(3/3)

- Eight operators, which take no operands

| operator | notation | action |
|---|---|---|
| j | MovD | D := [D] |
| i | Jmp | C := [D] |
| * | Rot | A,[D] := rotr([D]) |
| p | Opr | A,[D] := crz(A,[D]) |
| < | Out | putchar(A) |
| / | In | A := getchar() |
| v | Hlt | halt |
| o | Nop | no operation |
| other printable | Nop | no operation |
| non-printable | — | infinite loop |

- Codes are rewritten after step execution
- C and D registers are incremented after step execution



## Example of execution

- "HEllo WORld"

```
     A       C  D adr     data          opr
0000000000t 0  0   0 0000001111t (    j MovD
0000000000t 1 41   1 0000002021t =    p Opr
1111112211t 2 42   2 0000002020t <    p Opr
0000002200t 3 43   3 0000010120t `    < Out

                  41 0000002211t L    / In
                  42 0000011110t x    o Nop
                  43 0000002002t 8    < Out
```



## Repeated access data on memory(1/2)

- No load nor save operation. We have only
  - Rot: A,[D] := rotr([D])
  - Opr: A,[D] := crz(A,[D])
- Problem: Hard to access data repeatedly
- A solution: preparing codes so that D register loops in data area (D register is incremented in similar to C register)



## Repeated access data on memory(2/2)

- Example: implementation of pseudo-code

Pseudo code

```
label operation
      Rot CON2
      Opr X
      Rot CON2
      Opr X

CON2: 2222222222t
   Y:
   X:
```

A := X (nondestructive load of X)

```
  label data/opr
C       Rot
        Nop
        Opr
        MovD
        Rot
        Nop
        Opr
        MovD

D CON2: 2222222222t
     Y:
     X:
        CON2 - 1
```



## Trit-wise operations

Fact: Any trit-wise operations are functionally composable from 0, 1, 2 by $crz$

Proof: By exhaustive search

Example: $inc(\mathrm{X},\mathrm{Y})$ increment

| Y\X | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 2 | 0 |
| 1 | 1 | 2 | 0 |
| 2 | 0 | 1 | 2 |

$inc(\mathrm{X},\mathrm{Y}) = crz(crz(crz(crz(2,\mathrm{X}),0),\mathrm{Y}),crz(2,\mathrm{X}))$

- Note: It does not guarantee the existence of codes, since Opr is destructive operation

## Generating special constants(1/3)

- CON1: 1111111111t
  Precondition: variable CON1 has no 1s

```
Rot CON1
Opr CON1
```

- CONO: 0000000000t

```
Rot CON1
Opr CONO
Opr CONO
Opr CONO
```



## Generating special constants(2/3)

- CON2: 2222222222t
  Precondition: TMP has a pattern ...20...
  (... parts contain no 2s)

```
Rot CON1
Opr CON2
Opr CON2
```

  Now variable CON2 has been set 1111111111t

```
Rot TMP
Opr CON2
```

  repeat 10 times in total



## Generating special constants(3/3)

- P20: 2222222220t
  Precondition: TMP has a pattern .....2..
  (each . is not 2) and P20 = CON2

```
Rot TMP
Rot TMP
Opr P20
```

- Possible initial code for data area

```
adr label opr  data
85  TMP   In   0000011200t
88  CON1  Nop  0000002202t
91        Hlt  84
```



## Zero clear

- X := 0

```
Rot CON1
Opr X
Opr X
Opr X
```



## Nondestructive data copy

- Y := X using Z for temporal area

```
Rot CON1
Opr Z
Opr Z
Opr Y
Opr Y
Rot CON2
Opr X
Rot CON2
Opr X
Opr Z
Opr Y
```



## Successor function: X := X+1

- Precondition:
  Y = 2222222220t and
  Z contains no 2s

```
Rot CON1     Opr Y     Opr Y
Opr Z        Opr X     Rot CON2
Rot CON2     Opr Z     Opr Y
Opr X        Opr Z     Rot X
Opr Z        Opr Y
```

  repeat 10 times in total

## Generating arbitral data(1/2)

- Trit-wise operation $f(\mathtt{X},\mathtt{Y}) = crz(2, crz(\mathtt{X},\mathtt{Y}))$

| Y\X | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 2 | 0 | 0 |
| 1 | 2 | 0 | 1 |
| 2 | 1 | 1 | 2 |

- Precondition: right-most trit of `X` be `1`

```
Opr X
Rot CON2
Opr X
```

makes right-most trit of `X` to
`0` if `A = 2222222221t` (=P21)
`1` if `A = 2222222222t` (=CON2)
`2` if `A = 2222222220t` (=P20)
without changing the other trits



## Generating arbitral data(2/2)

- Generation of arbitral data is possible from `X = 1111111111t` by previous code and "`Rot X`"
- Loading `P20`

```
Rot CON0
Opr P2X
```

- Loading `P21`

```
Rot CON1
Opr P2X
```

where `P2X` is `2222222220t` or `2222222221`



## This talk

- Explanation of Malbolge
- Overview of Programming technique
  - Bootstrapping
  - Low level assembly language
  - High level assembly language




## Low level assembly language

- Designed for writing loop programs.
- Syntax
  - Labeled sequence of `U_JMP`, `U_ROT`, `R_ROT`, `U_MOV_PC`, `R_MOV_PC` and so on, and Flags
- Semantics: ternary virtual machine
  - Registers: A and PC
  - Need to execute `R_hoge` after executing operator `U_hoge`
  - Variables must be placed below and near the operation. All operation between the operation and the variable are skipped unexecuted.
  - Flags act as flip-flopped `U_MOV_PC`



- Example of low level assembly code for a pseudo code

```
Rot X
Opr X

X:  30527
```

Pseudo code

```
L1:     R_ROT
        U_ROT X
ENTRY:  U_ROT X
X:      30537
        FLAG1
                L1
        R_OPR
        END
```

Low level assembly code



## Character-replacement analysis(1/3)

- Replacement by `xlat2[]`

```
const char xlat2[] =
 "5z]&gqtyfr$(we4{WP)H-Zn,[%\\3dL+Q;>U!pJS72FhOA1C"
 "B6v^=I_0/8|jsb9m<.TVac`uY*MK'X~xDl}REokN:#?G\"i@";
for (;;) {
    ... OMITTED ...
  mem[c] = xlat2[mem[c] - 33];
}
```

- Periodic table

| ID | period | sequence |
|---|---|---|
| #1 | 2 | F J |
| #2 | 4 | * r } i |
| #3 | 5 | ) f ` < 3 |
| #4 | 6 | % g u o x : |
| #5 | 9 | 2 P B > L 0 C U I 2 |
| #6 | 68 | ! 5 - w N 1 W 0 { G S ~ 9 [ ··· |

## Character-replacement analysis(2/3)

- Operators for sequence with period two

| adr | op. F | op. J |
|---|---|---|
| 7 | — | Hlt |
| 11 | Hlt | — |
| 24 | — | Jmp |
| 25 | — | Out |
| 28 | Jmp | — |
| 29 | Out | — |
| 43 | — | In |
| 47 | In | — |

| adr | op. F | op. J |
|---|---|---|
| 59 | — | Rot |
| 60 | — | MovD |
| 63 | Rot | — |
| 64 | MovD | — |
| 82 | — | Opr |
| 86 | Opr | — |
| 88 | — | Nop |
| 92 | Nop | — |

  - Address are in mod 94
  - Symbol — represents non-operator that works as `Nop`
  - e.g. J at 59 and F at 63 are `Rot/Nop`



## Character-replacement analysis(3/3)

- `Jmp` is not replaced by execution, instead the operater at one earlier location of the new location is replaced
- Character corresponding to `SNop` exists for every location
  - Stable Nop (`SNop`): no operation appears in the sequence except `Nop` or `Nop`



## Implementation of L Ass(1/3)

- Basic idea
  - Construct operation units each of which consists of an operation with cycling period two
  - Use `D` register as program counter PC



## Implementation of L Ass(2/3)

- Operation units
  - Prepare a module for each operator
  - Example of unit for `Rot`

```
        SNop
        SNop
U_ROT:  SNop
R_ROT:  Rot/Nop
        Jmp
```



## Implementation of L Ass(3/3)

- Example of translation

```
L1:     R_ROT
        U_ROT X
ENTRY:  U_ROT X
X:      30537
        FLAG1
                L1
        R_OPR
        END
```

L Ass code

```
L1:     R_ROT
        U_ROT-1
ENTRY:  U_ROT
X:      30537
        FLAG1
        L1-1
        R_OPR
        R_HLT
```

Data for D register



## Programs written in L Ass

- rot13 program (33KB)

```
% ./malbolge rot13.mb
abcdefg
nopqrst
```

- 99 bottles of beer on the web

## This talk

- Explanation of Malbolge
- Overview of Programming technique
  - Bootstrapping
  - Low level assembly language
  - High level assembly language




## High level assembly language

- Designed for relaxing restrictions of L Ass
- Syntax and semantics
  - Labelled sequence of `INC X`, `DEC X`, `MOV X Y`, `BRANCH X Y`, `INPUT X`, `OUTPUT X` and `STOP`
- Example

```
L1:     DEC X
        INC Y
ENTRY:  OUTPUT Y
        BRANXH X L1
        STOP
X:      3
Y:      64
```



## Basic ide of implementing H Ass

- Transform each operation into a sequence of addresses

```
L1:     DEC X
        INC Y
ENTRY:  OUTPUT Y
        BRANCH X L1
        STOP
X:      3
Y:      64
```

```
L1:     DEC
        X
        INC
        Y
ENTRY:  OUTPUT
        Y
        BRANCH
        X
        L1
        STOP
X:      3
Y:      64
```

- Prepare a module for each operation, in which it copies the succeeding address into internal variable before processing