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 one down and pass it around, 98 bottles of beer on the wall.
...
2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.
1 bottle of beer on the wall, 1 bottle of beer.
Take one 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!

99 bottles of beer (3/3)

- Comments for Malbolge

  - http://members.tripod.com/rrusnery/weird.html

Esoteric Programming Languages

- Language designed to be HARD to read and write
  - INTERCAL [Woods, Lyon 1972]
  - BrainF*** [Müller 1993]
  - Befunge [Pressy 1997]
  - Malbolge [Olmstead 1998]

- Proposed as jokes

References

http://members.tripod.com/rrusnery/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
    \[
    A, [D] := \text{crz}(A, [D])
    \]
    
    \[
    [D] \backslash A \begin{array}{c}
    0 \\
    1 \\
    2 \\
    \end{array}
    \begin{array}{c}
    0 \\
    1 \\
    2 \\
    \end{array}
    \begin{array}{c}
    2 \\
    1 \\
    2 \\
    \end{array}
    \]
  - 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

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

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

Example of Malbolge program

- “HELlo WORld” program by Cooke.

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

Semantics of Malbolge (1/3)

- Malbolge machine
  - Address area = One word = ten trit
  - Codes and data are stored in memory
  - Three registers
    - A: accumulator
    - C: code pointer
    - D: data pointer

This talk

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

- Execution

```c
const char xlat2[] =
"5z\&gqtyfr$(we4{WP)H-Zn,\[%\3dL+Q;>U!pJS72FhDAIC"
"86x_i0/Dy\jebM.AVaye\y\H(K\x'\zD)\Re\o\x88\7D\'l8";
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 '<': x = getc( stdin ); a = x; break;
    case '/': x = putc( a, stdout ); break;
    case 'v': return;
  }
  mem[c] = xlat2[mem[c] - 33];
  if ( c == 59048 ) c = 0; else c++;
  if ( d == 59048 ) d = 0; else d++;
}
```

- Eight operators, which take no operands

<table>
<thead>
<tr>
<th>Operator</th>
<th>Notation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>MovD</td>
<td>D := [D]</td>
</tr>
<tr>
<td>i</td>
<td>Jmp</td>
<td>C := [D]</td>
</tr>
<tr>
<td>*</td>
<td>Rot</td>
<td>A, [D] := rotr([D])</td>
</tr>
<tr>
<td>p</td>
<td>Opr</td>
<td>A, [D] := crz(A, [D])</td>
</tr>
<tr>
<td>&lt;</td>
<td>Out</td>
<td>x := putc(x, stdout); break;</td>
</tr>
<tr>
<td>/</td>
<td>In</td>
<td>A := getchar()</td>
</tr>
<tr>
<td>v</td>
<td>Hlt</td>
<td>halt</td>
</tr>
</tbody>
</table>

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

Repeated access data on memory

- 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)

Example of execution

```
A  C  D  adr  data  opr
0000000000t 0 0 0 0000001110t  j MovD
0000000000t 1 41 1 0000002021t  = p Opr
1111112211t 2 42 2 0000000202t  < 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

- Example: implementation of pseudo-code

```c
Pseudo code

```ex
```

Trit-wise operations

- Fact: Any trit-wise operations are functionally composable from 0, 1, 2 by crz
- Proof: By exhaustive search
- Example: inc(x, y) increment
- Y \( \times \) increment
  - \( \begin{array}{ccc}
  Y \times X & 0 & 1 & 2 \\
  0 & 1 & 2 & 0 \
  1 & 1 & 2 & 0 
  \end{array} \)
- 2 \( \times \) 0 \( \times \) 1

- 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

- **CON0**: 0000000000t
  
  Rotate CON1 Opr CON1

Generating special constants (2/3)

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

  Rotate CON1 Opr CON2 Opr CON2

  Now variable CON2 has been set 1111111111t

  Rotate CON1 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

  Rotate TMP Opr P20

Possible initial code for data area

<table>
<thead>
<tr>
<th>adr</th>
<th>label</th>
<th>opr</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>TMIn</td>
<td>0000011200t</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>CON1</td>
<td>Nop 0000002202t</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>Hlt</td>
<td>84</td>
<td></td>
</tr>
</tbody>
</table>

Nondestructive data copy

- **X := 0**

  Rotate CON1 Opr X

Successor function: **X := X+1**

- Precondition:
  
  \[ Y = 2222222220t \] and
  
  \[ Z \text{ contains no 2s} \]

  Rotate CON1 Opr Y Opr Y

  Rotate CON2 Opr Z Opr X

  Rotate CON2 Opr Z Opr Y

  Rotate CON2 Opr X Opr Z

  Rotate CON2 Opr X Opr Y

  Rotate CON2 Opr X Opr Y

  Repeat 10 times in total
Generating arbitral data (1/2)

- Trit-wise operation \( f(x, y) = crz(2, crz(x, y)) \)

\[
\begin{array}{c|c|c}
Y & 0 & 1 & 2 \\
\hline
0 & 2 & 0 & 0 \\
1 & 2 & 0 & 1 \\
2 & 1 & 1 & 2 \\
\end{array}
\]

- Precondition: right-most trit of \( x \) be 1

\[
\begin{align*}
\text{Opr X} \\
\text{Rot CON2} \\
\text{Opr X}
\end{align*}
\]

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

\[
\begin{align*}
\text{Rot CON0} \\
\text{Opr P2X}
\end{align*}
\]

- Loading P21

\[
\begin{align*}
\text{Rot CON1} \\
\text{Opr P2X}
\end{align*}
\]

where P2X is 2222222220t or 2222222221

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 un-executed.
  - Flags act as flip-flopped U_MOV_PC

Example of low level assembly code for a pseudo code

\[
\begin{align*}
\text{L1:} & \quad \text{R_ROT} \\
\text{U_ROT X} \\
\text{ENTRY:} & \quad \text{U_ROT X} \\
\text{X:} & \quad 30537 \\
\text{FLAG1} & \quad \text{R_OPR} \\
\text{END } & \quad \text{L1}
\end{align*}
\]

Character-replacement analysis (1/3)

- Replacement by xlat2[]

```c
const char xlat2[] =
"S2\$g\xyz"5z&gqtyfr$(we4{WP)H-Zn,\%3dL+Q;\textbf{>U!pJ\textbf{S7}2\textbf{F}hOA1C";
for (;;) {
... OMITTED ...
} \textbf{>E}Hed
mem[c] = xlat2[mem[c] - 33];
```

- Periodic table

<table>
<thead>
<tr>
<th>ID</th>
<th>Period</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>2 F J</td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>4 * r } i</td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>5 ) f ' &lt; 3</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>6 % g u o x :</td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>9 2 P B &gt; L O C U I 2</td>
<td></td>
</tr>
<tr>
<td>#6</td>
<td>68 ! 5 - w N I W 0 { G S ~ 9 [ ...</td>
<td></td>
</tr>
</tbody>
</table>

This talk

- Explanation of Malbolge
- Overview of Programming technique
  - Bootstrapping
  - Low level assembly language
  - High level assembly language
### Character-replacement analysis (2/3)

- Operators for sequence with period two

<table>
<thead>
<tr>
<th>adr</th>
<th>op.</th>
<th>F</th>
<th>Op.</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>Hit</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Hlt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Jmp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Out</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Jmp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Out</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>In</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>In</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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 operator 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

<table>
<thead>
<tr>
<th>SNop</th>
<th>SNop</th>
<th>U_ROT: SNop</th>
<th>R_ROT: Rot/Nop</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SNop</td>
<td>U_ROT X</td>
<td>R_ROT X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENTRY: U_ROT X</td>
<td>ENTRY: U_ROT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X: 30537</td>
<td>X: 30537</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLAG1</td>
<td>FLAG1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DPR</td>
<td>R_DPR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>END</td>
<td>END</td>
</tr>
</tbody>
</table>

### Implementation of L Ass (3/3)

- **Example of translation**

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

### Programs written in L Ass

- **rot13 program (33KB)**

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

- **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
```

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