社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

難読プログラミング言語Malbolgeにおけるプログラム構成手法

飯澤 恒† 坂部俊樹 †† 酒井正彦 †† 草刈圭一朗 †† 西田直樹 ††

†, †† 名古屋大学大学院 情報科学研究科 〒 464–8603 愛知県名古屋市千種区不老町
E-mail: †iizawa@sakabe.i.is.nagoya-u.ac.jp, ††{sakabe,sakai,kusakari,nishida}@is.nagoya-u.ac.jp

あらまし プログラミング言語 Malbolge は，意図的に言語仕様を難解に設計し，その言語上でのプログラムの作成や解読を困難にすることを目的とした，難読（難解）プログラミング言語である．本研究では，ソフトウェア保護技術であるプログラムの難読化に応用することを目的として，Malbolge 上で高度な機能を実現するためのプログラミングの指針を示す．

キーワード プログラミング言語, ソフトウェア保護, 難読化, Malbolge

Programming Method in Obfuscated Language Malbolge

Hisashi IIZAWA†, Toshiki SAKABE††, Masahiko SAKAI††,

Keiichirou KUSAKARI††, and Naoki NISHIDA††

†, †† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya City, Aichi, 464–8603 Japan
E-mail: †iizawa@sakabe.i.is.nagoya-u.ac.jp, ††{sakabe,sakai,kusakari,nishida}@is.nagoya-u.ac.jp

Abstract Malbolge is an obfuscated (esoteric) programming language, which is designed to be difficult to program in. In this paper, we propose a guide for programming in Malbolge for the purpose of the application for obfuscation in software protection.

Key words Programming Language, Software Protection, Obfuscation, Malbolge

1. ま え が き

作成したプログラムを不特定多数のユーザーに配布する場合，システムの安全性の確保や知的財産権保護等を目的として，プログラムの内部が解析困難であることが求められる場合がある．そのためには，通常の手法でプログラムを作成した後，それが解析困難となるように何らかの方法でプログラムを変換するという方法が取られる．この等価変換を難読化と呼ぶ [1], [4].

プログラミング言語の中には，難読（難解）プログラミング言語と呼ばれる特殊な言語が存在する．これらの言語は設計思想が通常の言語とは異なり，意図的にその言語上でのプログラムの作成や理解が困難になるように設計されている．難読言語で作成されたプログラムは読解が困難であるため，難読として優れていると言える一方，プログラミングが困難であるという問題を持っている．

本論文で対象とするのは，Malbolge という言語である [2], [3]. これは “人類が設計しうる最も邪悪な言語” と称され，難読言語の中でも特に難解な言語として知られている．以下に，Malbolge のサンプルプログラムとして “Hello World!” のプログラムを示す．

```
(=<;:9]~6Z:z2VU/.R2+*)Mn&%I#"!E}|Bzy?wv;zsKqpot3EqpiAm
lONihKs_d]\[‘}A0\>>Y<:V97‘R4nON1/EJIHGFEDCB‘:9]=654XE
```

Malbolge 上でのプログラミングについての先進研究 [3] は存在するが，複雑なプログラムの作成を可能とするまでには至っておらず，Malbolge で複雑なプログラムを作成する手法は一般には知られていない．

Malbolge 上でプログラミングするにあたっての問題点は以下のものが挙げられる．

- 各命令が限定的な機能しか持たず，またその機能もプログラミングに不適であること．
- プログラムロード時のデータの初期値が著しく制限されること．
- 強制的な命令の動的書き換えにより，同じ命令列を繰り返し実行することが困難であること．

本論文では，これらの問題点について解決策を示した後，それらを踏まえて，Malbolge 上で一般的なプログラムを作成する指針を示す．

表 1　演算子 op
Table 1　Operator 'op'

		X_i		
		0	1	2
Y_i	0	1	0	0
	1	1	0	2
	2	2	2	1

2. Malbolge の仕様

2.1 環　　境

Malbolge では，1 word は 10 trit すなわち 3 進数 10 桁で表現され，0 から 59048 までの値をとる．メモリ空間はコード／データ共有で，59049 word の領域を持つ．これを mem[0]〜mem[59048] と表現する．また，3 つのレジスタ（A,C,D）を持ち，A はアキュームレータレジスタ，C はコードポインタ，D はデータポインタである．

データに対する演算子として trit 演算 op(X,Y) を持つ．これは，2 つの入力値の各桁毎に，表 1 で表される演算を行う tritwise な 2 項演算子である．

例: op([0120120120], [0001112220]) → [1001022211]

プログラムコードは全て印字可能な文字（ASCII:33〜126）で構成され，ロード時に，全ての文字が命令として解釈可能な命令文字であることがチェックされる．

印字可能文字から他の印字可能文字への変換表，xlat1 と xlat2 がある．xlat1 は命令解釈に，xlat2 は命令文字置換に用いられる．各変換表は配列で以下のように定義されている．ともに全単射である．

```
const char xlat1[] =
  "+b(29e*j1VMEKLyC})8&m#~W>qxdRpOwkrUo[D7,XTcA\"lI"
  ".v%{gJh4G\\-=O@5‘_3i<?Z';FNQuY]szf$!BS/|t:Pn6^Ha";
const char xlat2[] =
  "5z]&gqtyfr$(we4{WP)H-Zn,[%\\3dL+Q;>U!pJS72FhOA1C"
  "B6v^=I_O/8|jsb9m<.TVac‘uY*MK'X~xDl}REokN:#?G\"i@";
```

この場合 xlat2 を用いた印字可能文字 c からの変換は xlat2[c-33] と記述される．

2.2 命　　令

Malbolge が持つ命令は以下の 8 つのみである（付録参照）．

- ‘i’ … INPUT：入力命令
- ‘j’ … LOAD_D：データレジスタ変更命令
- ‘*’ … ROTATE：右ローテート命令
- ‘p’ … OPR：演算命令
- ‘o’ … NOOP：無操作命令
- ‘v’ … HALT：停止命令
- ‘<’ … INPUT：入力命令
- ‘/’ … OUTPUT：出力命令

BRANCH/LOAD_D は，その時点での mem[D] の値をレジスタ C/D にセットする命令である．

ROTATE は mem[D] の値の右ローテートを計算し，OPR は op(A,mem[D]) を計算する．どちらも，計算結果は A と mem[D] にセットする．

INPUT/ OUTPUT は，標準入出力から 1 文字を入出力する命令である．

上記の各先頭の文字（i,j,*,･･･,/) は命令解釈後の文字を表しており，実行に際しては，mem[C] を解釈した結果がこれらの文字のいずれかである場合に，対応する命令が実行される．mem[C] の命令解釈は xlat1[(mem[C]+C-33)%94] で行われる．このとき，メモリ中の k 番地にある文字 mem[k] が命令文字であるとは，xlat1[mem[k]+k-33)%94] が {i,j,*,p,o,v,<,/} のいずれかであることをいう．

2.3 実行ステップ

1 ステップは以下のように実行される（付録参照）．

（1） mem[C] を調べ，印字可能文字で無い場合は暴走する．
（2） mem[C] を命令として解釈し，その結果に基づいて該当する命令を実行する．どの命令にも該当しなければ（命令文字でなければ）NOOP と同じとみなす．
（3） xlat2 を用いて mem[C] を別の文字へ置換する．
（4） C,D をインクリメントする．

これを停止するまで繰り返す．

3. Malbolge の問題点

Malbolge 上でプログラミングを行うに当たって，解決すべき問題点は主に以下の 3 点である．

a） 問 題 点 1

各命令の持つ機能が極めて限定的であり，またその機能もプログラミング言語として自然であるとは言い難い．そのため，複数の命令を組み合わせてより高度な機能を実現することが難しくなっている．

例えば，全てのデータ操作は ROTATE と OPR だけで行わなければならないが，高度な演算を実現しようにも，演算子 op からは数学的な特性といったものは見出しにくい．また，条件分岐を行う際には，判定対象となるメモリの値に応じて，mem[D] の値を異なるアドレス値にセットして BRANCH 命令で分岐させることになるが，その際の，判定して異なるアドレス値を導くという操作も，OPR や ROTATE の組み合わせで実現しなければならない．

b） 問 題 点 2

複雑な計算を行うためには，メモリを適切な値で初期化する必要があるが，Malbolge ではプログラムロード時のデータの初期値は著しく制限される．

仕様上，プログラムコードが正しくロードされるためには，プログラム中の全ての文字が命令文字でなければならないという制約(注1)がある．つまり，プログラム中の全ての文字が高々8 種類の命令文字によって構成されていなければならないという制約であり，これにより，プログラムロード時のメモリ 1word は，8 つの命令文字のいずれかによってしか初期化されない．

c） 問 題 点 3

仕様により，一度実行した命令文字は xlat2 によって別の文

(注1)：インタープリタ [2] にはバグがあり，印字可能でない文字をそのままメモリにロードしてしまうという脆弱性がある．

表 2　インクリメント用演算
Table 2　Operation for Increment

		X_i		
		0	1	2
Y_i	0	0	1	2
	1	1	2	0
	2	1	2	0

字へ置換される．このため，一旦実行した命令列は一時的にしろ全く別の文字列へと書き換えられるため，繰り返し同じ命令列を実行することが極めて困難になる．

4. 問題点 1 の解決

4.1 演算子 op の関数としての能力

OPR と ROTATE 命令は C 言語のビット演算とシフト演算に相当する．故に，op を関数合成することで他の様々な tritwise 関数を実現できるかどうかは，加算等の一般的な機能を実現するための重要なポイントである．

例えば，インクリメントを実現するためには，表 2 のような tritwise 関数が必要となる．この場合，$Y = [0000000001]$ としてこの関数を適用すれば X の最下位桁がインクリメントされる．

まず，op を関数合成することで任意の tritwise 関数を作成できるかどうかを調査した．その結果，任意の tritwise 関数が深さ 8 までの合成により実現できることを確認した．以下は表 2 の関数を実現する合成の一例である．

op(op(op(X,C0), op(C2, op(Y,C0))), op(C2,X))

ただし，Ci($i = 0, 1, 2$) はそれぞれ各桁が全て i である定数である．

4.2 データモジュール

高度な演算を実現するためには，データや定数に繰り返しアクセスすることが必要となる．これを可能とするため，対象となるデータと演算に必要な定数，アドレスを一つにまとめた，**データモジュール**というデータ構造を考案した（図 1）．

この方式を用いることで，NOOP や LOAD_D 命令を適当な回数実行することにより，データモジュール上の任意の位置に D レジスタのポインタを移動させることができる．例えば，D レジスタが CON1 を指している状態で MEM2 へ D レジスタをシークさせたい場合は，NOOP,NOOP,LOAD_D,NOOP と実行すればよい．これにより，データに対して繰り返し演算を行うことが容易となった．

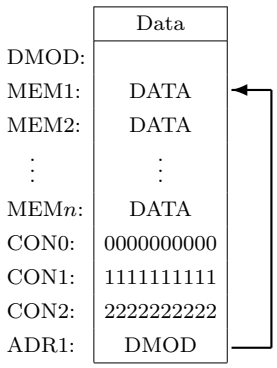

図 1　データモジュール
Fig. 1　Data Module

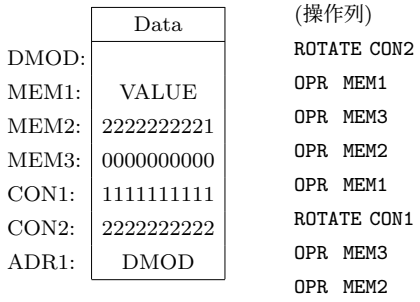

図 2　インクリメント用データモジュール
Fig. 2　Data Module for Increment

例として，op(op(C1,MEM2), op(C2,MEM1)) を実現するには，

```
ROTATE CON2
OPR MEM1
ROTATE CON1
OPR MEM2
OPR MEM1
```

のように実行すればよい．ここで，OPR MEM1 とは，MEM1 にシークした後 OPR を実行することを意味する．このような命令列をデータモジュールの操作列と呼ぶ．

ただし，4.1 で得られた op の関数合成は必ずしもデータモジュール上で実現しやすいものばかりではないため，データモジュール上でインクリメント等の具体的な機能を実装するにあたっては，あるデータモジュール上での操作列がどのような演算効果を持つかを虱潰し探索することで，必要な演算を実現する．

例として，インクリメント用のデータモジュールと，その操作列の一部を図 2 に示す．この操作列は，MEM1 の最下位 1 trit のインクリメントを行うもので，これを桁上げを考慮しつつ全桁について行えば，インクリメントが実現できる．

4.3 具体的な機能実現

データモジュールを用いることで，以下の機能が実現できることを確認した．

- インクリメント／デクリメント
- 定数生成
- 条件分岐
- アドレス指定のデータコピー

ただし，データコピーには，アドレス指定でデータにアクセスできるように拡張したデータモジュールを用いる．

5. 問題点 2 の解決

データモジュールにより，定数生成すなわち，命令列の実行によって特定のメモリあるいは A レジスタ上に任意の値を作り

No.1（周期=2）	F → J → F
No.2（周期=4）	* → r → } → i → *
No.3（周期=5）	) → f → ‘ → < → 3 →)
No.4（周期=6）	% → g → u → o → x → : → %
No.5（周期=9）	2 → P → B → > → L → 0 → C → U → I → 2
No.6（周期=68）	! → 5 → - → w → N → 1 → W → 0 → { → G → S → (中略) → ’ → t → E → p → D → !

表 3　xlat2 の変換サイクル
Table 3　Cycles of xlat2

出すことが可能となった．また，4.3 の他の機能により，生成した値を指定したアドレスへコピーすることも可能となる．

これにより，(1) 定数を生成 (2) 指定したアドレスへコピー (3) アドレスをインクリメント という操作を繰り返すことで，希望のメモリ領域を自由な値で初期化することが可能となった．

ただし，定数生成用データモジュール自身の構成にはこの方法は使えないため，このデータモジュールは次のように構成する．まず，データモジュールの予定位置を適当に決め，少なくとも ADR1（図 1 参照）だけは予め正しい値に初期化できるようにする．このようにすることによって，最低限データモジュールとしての繰り返しアクセスは可能となる．後は，他のメモリの各初期値をうまく利用し，[0000000001] のような最下位桁だけ trit 値が異なるようなデータを作り出し，それを元に他のメモリを一桁ずつセットしてゆく．実際にこの方法によって，制限の範囲内で定数生成用データモジュールが構成可能であることが確認済みである．

6. 問題点 3 の解決

以下では，プログラムが反復実行可能であることを，そのプログラムは**ループ耐性**を持つと呼ぶ．ループ耐性を持つプログラムを作成するために，主に 2 つのアイデアを用いた．

一つは，変換表 xlat2 の周期性である．xlat2 の変換は全単射であり，印字可能文字は有限であるため，任意の文字から変換を繰り返し行えばいつかは必ず元の文字に戻ってくる（図 3）．

もう一つは，BRANCH 命令の不変性である．xlat2 による置換は命令実行後の mem[C] に対して行われるため，BRANCH 命令に関しては実行後にその命令文字が置換される心配が無い．

6.1 限定的ループ耐性

比較的小規模なプログラムに対して，ループ耐性を実現する手法は [3] にて原案が示されている．

これには前述のアイデアの両方を用いる．xlat2 の変換サイクルの中には，特定のメモリ位置において，サイクル中に一度だけ特定の命令を実行し，それ以外の部分では何の命令も実行されないという特殊なサイクルが存在する．これを**単一命令サイクル**と呼ぶ．

例えば，mem[8]=‘}’ にて ‘}’ は ROTATE に解釈されるが，‘}’ からの変換サイクル（→ i → * → r → }）において，命令文字は ‘}’ のみであり，他の文字は実行時には NOOP と同じと見なされる．

単一命令サイクルがあると，特定の命令の実行にループ耐性を持たせたい場合にとても有用である．なぜなら，その命令の実行の後，そのサイクルの周期-1 回だけ繰り返し実行を行えば，その間に副作用を起こすことなく再び元の命令文字に復元されるからである．

命令列のループ耐性を実現するためには大量の単一命令サイクルが必要であるが，単一命令サイクルの存在頻度は決して高くなく，メモリ内に散らばって存在する（特定の文字がどの命令に解釈されるかは位置に依存するため）．

このため本手法では，単一命令サイクルをメモリ中に散らばった状態で確保し，それぞれの直後に BRANCH 命令を配置し，飛び石状に各サイクルをつなぐ．このように命令を配置することで，一旦命令列を実行した後，各サイクルの周期の回数だけ実行を繰り返すことで，再び元の命令列を得ることができ，また，BRANCH 命令自体は不変であるため，ループ耐性が実現される．

例えば，命令列 INPUT, OPR, ROTATE では，それぞれ周期 2 の単一命令サイクルがアドレス 25,82,59 に存在するので，

```
25: INPUT
26: BRANCH (to 82)
      ⋮
59: ROTATE
60: BRANCH (to ···)
      ⋮
82: OPR
83: BRANCH (to 59)
```

のように行う．この場合，周期は 2 なので，この命令列の実行すると 25,59,82 の命令文字は非命令文字となり，もう一度実行すると命令列は復元される．

6.2 一般的ループ耐性

6.1 の方法では，小規模なプログラムにループ耐性を持たせることはできても，大規模なプログラムには適用困難である．これを解決するため，**基本モジュール方式**を考案した（図 3）．

これは，基本機能を実現するモジュールを予め 6.1 の方法でループ耐性を持たせた状態で作成しておき，それらをサブルーチンコールのように BRANCH 命令で呼び出すという方法である．必要な基本機能を全てモジュールとして実現しておき，メインルーチンはこれらのモジュールを呼び出すだけとする．ここで，基本モジュールは CPU の命令のような役割を果たし，メイン

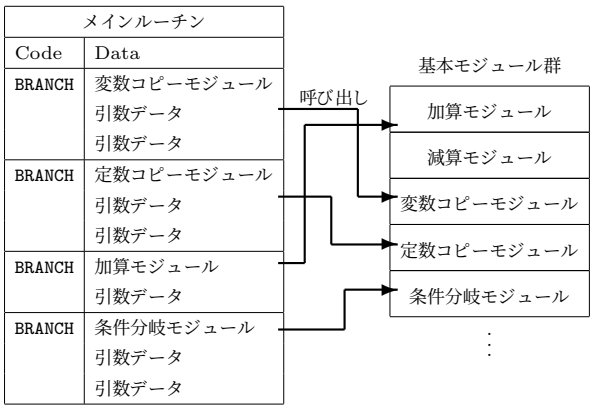

図 3　基本モジュール方式
Fig. 3　Basic Module System

ルーチンはアセンブリ言語のようなものになる．

呼び出したいモジュールのアドレスとその引数をまとめたものをデータ列として連ねておく．BRANCH 命令でそのモジュールに制御を移すと，モジュールは引数データを読み取って加算や代入などの演算を行い，処理が終わったら，予めメモリ中に保持されている C,D レジスタの値を元に，サブルーチンコールから復帰する．この操作をデータ列に従って次々と行うことで，一般的なプログラムを実現することができる．

この手法では，メインルーチンのコードは BRANCH 命令のみで構成されるため，前述の 2 つ目のアイデアにより，メインルーチンにもループ耐性を持たせることができる．

7. 一般的なプログラムの構成

ここまでに示してきた手法を用いて，Malbolge 上で一般的なプログラムを構成する手法を示す．

7.1 基本モジュールの設計

基本モジュール方式で用いられる，各種モジュールは予め作成しておく．ここで，各コード・データを構成する word 値に制限は無いものとして設計する．一度作成した基本モジュールは他のプログラムでも再利用可能である．

基本モジュールは 6.1 の手法でループ耐性を持たせるので，その設計は以下のように行う．

（1） 4.2 の手法で，モジュールの機能を実現するデータモジュールとそれを操作する操作列を設計する．ただし，この時点ではこのモジュールはループ耐性を持たない．

（2） 設計した操作列中の各命令について，その命令を実現する単一命令サイクルを割り当て，そしてそれらが直後の BRANCH で繋がるように，必要なアドレス情報をデータモジュールに挿入する．

（3） 置換された命令列を復元する際の実行に必要なダミーのデータ列を設計する．

7.2 メインルーチンの設計

各基本モジュールの呼び出しを記述することで，目的の機能を実現するメインルーチンを設計する（6.2 参照）．ここでも，各コード・データを構成する word 値に制限は無いものとして設計する．

7.3 実行可能なプログラムの作成

実行可能な Malbolge のプログラムコードを作成する．ここまでで設計した各種コード・データは，初期値制限を無視した word 列となっているため，これをそのままプログラムコードとすることはできない．このため，5. の手法を用いて，これらの word 列をメモリ上に展開する．

以下の順で作業を行うようにプログラムを作成する．

（1） 定数生成用のデータモジュールを構成する．

（2） (1) を用いて，領域初期化に必要なデータコピー等のモジュールをメモリ上に構成する．

（3） (1),(2) を用いて，各基本モジュールのコード・データをメモリ上に展開する．

（4） (1),(2) を用いて，メインルーチンのコード・データをメモリ上に展開する．

（5） 残ったメモリ領域を，自由に利用可能な領域としてゼロクリアする．

（6） メインルーチンに制御を移す．

7.4 自動化についての考察

上記のプログラムの構成手法は，人の手で行うには極めて繁雑である．よって，これらの作業を自動的に行うコンパイラが実現できるかということが重要となる．ここでのコンパイラとは，何らかの他言語を入力とし，それを実現する Malbolge のプログラムを出力するものを指す．

コンパイラを実現するためには，上記の各手順を辿る必要があるが，必ずしも全ての処理をコンパイラで行う必要は無い．なぜなら，上記手順のうち,「メインルーチンの設計」と「実行可能なプログラムの作成」の (4) 以外の処理については，入力プログラムに関わらず固定であるため，予め人の手でプログラムを作成しておき，コンパイル時にそれをリンクするだけでよいからである．

よって，コンパイラ側で実現しなければならない処理は以下の 2 つのみであり，これらは自動化可能であると思われる．

- 入力プログラムを解釈し，それを実現するメインルーチンを構成する．
- メインルーチンのコード・データをメモリ上に展開するための，定数生成モジュールの操作列を生成する．

8. ま と め

本稿では，難読化プログラミング言語である Malbolge を取り上げ，この Malbolge 上でプログラミングを行う上での各問題点について解決法を示し，一般的なプログラムを構成する手法を示した．

今後の課題としては，コンパイラを実装することと，それによって得られたプログラムの難読化の強度，すなわち逆変換の困難さについて，評価を行う必要があると考える．ただし，明らかに，基本モジュール方式での実装は難読化の強度を著しく低下させると考えられるため，何らかの対応策が必要であると思われる．

謝辞 本研究は一部，名古屋大学 21 世紀 COE プログラム（社会情報基盤のための音声・映像の知的統合）の補助を受けている．

文 献

[1] 門田暁人, 高田義広, 鳥居宏次, “プログラムの難読化法の実験的評価, ” 情処学ソフトウェア工学研報, vol.96, no.32, pp.33–40, Mar.1996.

[2] Malbolge: Programming from Hell (archived)
URL: http://web.archive.org/web/20010613070437
/www.mines.edu/students/b/bolmstea/malbolge/

[3] Programming in Malbolge
URL: http://www.lscheffer.com/malbolge.html

[4] ソフトウェアプロテクション
URL: http://se.aist-nara.ac.jp/themes/protection/

付 録

1. Malbolge インタープリタ 実行部

Malbolge インタープリタにおいて，実行部は以下のように

C 言語で比較的簡潔に記述される.

```
void exec( unsigned short *mem )
{
  unsigned short a = 0, c = 0, d = 0;
  int x;
  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] = op( a, mem[d] ); break;
      case '<': putc( a, stdout ); break;
      case '/':
        x = getc( stdin );
        if ( x == EOF ) a = 59048; else 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++;
  }
}
```