一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2012-37(2012-11)

三値関数を実現するMalbolge命令列の発見のための SATエンコーディング

安藤 聡† 酒井 正彦†† 坂部 俊樹†† 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: †ando@sakabe.i.is.nagoya-u.ac.jp, ††{sakai,sakabe,kusakari,nishida}@is.nagoya-u.ac.jp

あらまし Malbolgeは最も難解なプログラミング言語として知られている．低級アセンブリ言語の開発によりMalbolgeのループプログラムの作成が可能になったものの，低級アセンブリ言語には通常のプログラミング言語が持つような演算命令がなく，Malbolge特有の演算を行う命令のみであるため，低級アセンブリ言語でのプログラミングにも困難が伴う．低級アセンブリプログラミングにおいては，目的のプログラムに必要な二引数三値関数を実現するMalbolge特有の演算命令列の発見が求められる．しかしこれまで，そのような命令列を発見するための手法は網羅的な探索に限られており，低級アセンブリプログラミング上の制限となっていた．本稿では，二引数三値関数を実現するMalbolge命令列の発見にSATソルバを利用した手法を提案することで，この問題の解決を試みる．そのため，二引数三値関数を実現するMalbolge命令列を発見する問題を定式化し，その問題のSATエンコーディングを提案する．実験により，既存手法より高速に，かつ，長い命令列が発見できることが分かった．

キーワード 難解プログラミング言語，Malbolge，SATエンコーディング，三値関数

A SAT Encoding for Finding Operation Sequences of Malbolge that Implement Trit-wise Functions

Satoshi ANDO†, Masahiko SAKAI††, Toshiki SAKABE††,

Keiichirou KUSAKARI††, and Naoki NISHIDA††

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

Abstract Malbolge is known to be one of the most esoteric programming languages. Although it becomes possible to write programs in Malbolge language due to the development of the low-level assembly language, the programming in the low-level assembly language is difficult. This is because the arithmetic instructions of the low-level assembly language are not those of ordinary programming languages, but the same as the ones of Malbolge. To construct a program in low-level assembly language, it is necessary to find instruction sequences of Malbolge that implement binary trit-wise functions. For finding such instruction sequences, however, an inefficient exhaustive search is the only known method, which is one of limitations in developing programs in the low-level assembly language. In this paper, to solve this problem, we propose a method to find instruction sequences of Malbolge that implement binary trit-wise functions, which is more efficient due to the use of state-of-art SAT solvers. We formalize a problem that finds an instruction sequence of Malbolge that implements a given binary trit-wise function, and propose a SAT encoding for this problem. Experiments shows that our method is able to find instruction sequences faster than the existing method and hence essentially longer ones.

Key words Esoteric programming language, Malbolge, SAT encoding, Trit-wise function

1. はじめに

難解プログラミング言語は意図的にその言語でのプログラミングが困難になるように設計された言語である．そのような言語で書かれたプログラムは解読困難性を持つため，プログラムの改ざん防止や知的財産の保護に役立つと考えられている．

Malbolge [3] は難解言語の中でも特に難解として知られており，その難解性からプログラムの解読や変更だけでなくプログラミングも非常に困難である．これまでに低級アセンブリプログラムを Malbolge プログラムに変換する低級アセンブラが構築され [7], [8]，Malbolge のループプログラムの作成が不可能ではなくなった．しかしながら，低級アセンブリプログラムの開発は容易ではない．

低級アセンブリ言語には通常のプログラミング言語が持つような演算命令が存在せず，三進数の十桁からなるワードの右ローテート命令と，三進数の桁毎に特殊な二引数演算を行う命令の二つのみである．これは，プログラミングを困難にする理由の一つになっている．

低級アセンブリプログラムのプログラミングでは、まず演算命令の列をわかりやすく表現できる疑似命令で設計し，それを低級アセンブリプログラムにコーディングする方法が開発されている [1], [2], [7], [8], [11]．インクリメントなどの基本演算をプログラムする場合には，まず，二種類の演算命令を組み合わせていくつかの三値関数を実現し，得られた三値関数をさらに組み合わせることで基本演算を実現している．しかしながら必要となる二引数三値関数を実現する演算命令の組み合わせの発見は困難である．文献 [1] では探索プログラムを用いてこの解決を試みているが，探索に枝刈りを導入しているものの，探索時間が長く，実質的には 11 ステップ以上の命令系列を求めることは不可能である．

近年，ブール変数を持ち限量子を持たない論理式の充足可能性（SAT 問題）を判定する SAT ソルバの開発が進んでおり，NP 困難な問題を等価な SAT 問題に変換し SAT ソルバにより問題を解くという応用がなされてきている．本稿では，Malbolge の低級アセンブリプログラム開発の一部である，二引数三値関数を実現する命令系列の発見に SAT ソルバを利用する．そのため，この問題を等価な SAT 問題へ変換する手法を提案する．実験により，[1] の探索プログラムより高速に，かつ，長い命令系列が発見できることが分かった．

本稿の構成は次の通りである．2. 節では準備として，Malbolge，低級アセンブリ言語，疑似命令，並びに SAT 問題について説明する．3. 節では二引数三値関数を実現する命令系列を発見する問題を定式化し，SAT 問題へのコーディング法を提案する．4. 節では提案したコーディング法を用いた発見手法の実装を述べる．5. 節では既存手法との比較実験を行い，6. 節で全体をまとめる．

2. 準　　備

ここでは Malbolge，低級アセンブリ言語，疑似命令，並びに SAT 問題について本稿で必要な部分のみを説明する．Malbolge と低級アセンブリ言語，疑似命令についてのより詳細な説明は文献 [7], [8], [12] を参照されたい．

表 1　Malbolge の命令
Table 1　Malbolge Instructions

命令	表記	説明
i	Jmp	ジャンプ．C:=mem[D].
j	MovD	D レジスタの更新．D:=mem[D].
p	Opr	演算命令．A,mem[D]:=OP(A,mem[D]).
*	Rot	右ローテート．A,mem[D]:=ROTR(mem[D]).
/	Input	入力．A:=getchar().
<	Output	出力．putchar(A).
o	Nop	無操作．何も行わない．
v	Halt	終了．プログラムの実行を停止．
その他	Nop'	無操作．

表 2　trit 演算 op(x,y)
Table 2　Trit Operation op(x,y)

		x		
		0	1	2
y	0	1	0	0
	1	1	0	2
	2	2	2	1

2.1 Malbolge

Malbolge [3] は仮想機械上で動作する機械語であり，インタプリタによって意味が定められている．仮想機械は三つのレジスタ (A,C,D) とメモリ (mem) を持ち，値は三進数十桁 (10trits) で表現される．よって値は $0000000000t \sim 2222222222t$ となり，メモリのアドレス空間も mem[0]~mem[59048] で定義される．表 1 に Malbolge の命令を示す．命令 Opr の関数 OP(X,Y) は X と Y の各桁同士で表 2 の trit 演算 op を行う．変数 X の第 i 桁を X_i で表す．f を trit 演算とし，二つの引数 $X = X_nX_{n-1}\ldots X_1X_0t$，$Y = Y_nY_{n-1}\ldots Y_1Y_0t$ ($0 \leqq X_i, Y_i \leqq 2$, $0 \leqq i \leqq n$) に対して $f(X_n,Y_n)f(X_{n-1},Y_{n-1})\ldots f(X_1,Y_1)f(X_0,Y_0)t$ を計算する関数を **trit 演算 f から定まる二引数三値関数 F** と呼ぶ．

以下に命令 Opr で用いられる関数 OP の計算例を示す．

OP(0120120120t, 0001112222t) = 1001022212t

また，命令 Rot で用いられる関数 ROTR の計算例を以下に示す．

ROTR(0001112222t) = 2000111222t

2.2 低級アセンブリ言語

低級アセンブリ言語 [7] は Malbolge と同じメモリ空間を持つ仮想機械として定義される．レジスタは PC と A の 2 つを持ち，値は 10trits で表現される．

低級アセンブリプログラムは，メモリアドレスを表すラベルを付加可能な命令の列と変数定義から構成される．低級アセンブリ言語の命令を表 3 にまとめた．[*label*] は *label* が表すアドレスに格納された値を表しており，A は A レジスタの値を指す．

変数の初期値には，定数として十進数（0~59048）と三進数（0000000000t ~2222222222t）が利用でき，特別な定数として任意の定数を表す DUP が利用できる．

表 3　低級アセンブリ言語の命令
Table 3　Low-Level Assembly Language Instructions

命令	操作
ROT_UNIT *label*	A,[*label*] := ROTR([*label*])
REV_ROT	ROT 命令の復元
OPR_UNIT *label*	A,[*label*] := OP(A,[*label*])
REV_OPR	OPR 命令の復元
JMP_UNIT *label*	PC:= *label*
MOV_PC_UNIT *label*	PC:= *label*
REV_MOV_PC	MOV_PC 命令の復元
INPUT_UNIT	A:=getchr()
REV_INPUT	INPUT 命令の復元
OUTPUT_UNIT	putchr(A)
REV_OUTPUT	OUTPUT 命令の復元
FLAG *label*	FLAG が ON 場合：PC:= *label*
	FLAG が OFF の場合：何もしない
	命令実行後には FLAG がフリップされる
FLAG+1	FLAG がフリップされる (ON↔OFF)
END	終了

低級アセンブリプログラムは低級アセンブラ [7] を用いて Malbolge プログラムに変換できる．

2.3 疑似命令

疑似命令 [7] は，Malbolge の演算命令の表現をより単純化し，引数にメモリ名を指定できるようにしたものである．例えば，以下の疑似命令列を考える．

```
ROT  X
OPR  Y
```

これは以下の計算を意味する．

A,X := ROTR(X)
A,Y := OP(A,Y)

例えば，前者の場合，ROTR(X) の結果を A レジスタと変数 X に書き込む．

疑似命令の引数に定数として Con0(= 0000000000t)，Con1(= 1111111111t)，Con2(= 2222222222t) を利用できる．

2.4 SAT 問題

命題論理式とは，true か false のどちらかを取る変数（ブール変数）上で定義される論理式である．命題論理式が CNF であるとは，命題論理式がリテラルの選言で構成される節の連言の形をしていることである．リテラルとは変数かその否定である．論理式が充足可能（SAT）であるとは，その論理式全体を true とするような変数の割り当てが存在することである．存在しないならば，その論理式は充足不能（UNSAT）である．

SAT 問題とは，CNF が与えられたとき，その CNF が充足可能かどうかの決定問題である．

ES_1 節とは節中のリテラルのうち，ちょうど一つだけが true に解釈されるとき，かつその時に限り，その節が true に解釈される節のことである [5], [6]．リテラル $l_1, \ldots, l_m$ からなる ES_1 節を $\{l_1, \ldots, l_m\}_{ES_1}$ で表す．この ES_1 節は以下の論理式と等価である．

$$\bigvee_{i=1}^{m} l_i \ \wedge \bigwedge_{1 \leq i < j \leq m} \neg l_i \vee \neg l_j$$

3. 命令列の探索問題とコーディング

ここでは二引数三値関数を実現する疑似命令列の探索手法を提案する．3.1 節では探索を問題として定式化する．3.2 節では問題を SAT 問題にコーディングする際に用いるブール変数について定め，3.3 節ではコーディング法について述べる．

3.1 疑似命令列存在問題

ある二引数三値関数 F について，疑似命令列 I が $F(X,Y)$ を**実現する**とは，疑似命令列 I を実行した直後の A レジスタの値が $F(X,Y)$ となることである．

二引数三値関数に対する疑似命令列存在問題を次のように定義する．

インスタンス： trit 演算 f，LENG，VNUM．

解： 命令列長が LENG 以下で，変数 $Z^0, Z^1, \ldots, Z^{\mathrm{VNUM}+1}$ のみを使用し，f から定まる F について，$F(Z^0, Z^1)$ を実現し以下の条件をすべて満たす疑似命令列が存在するか？

条件 [1] ROT の引数は定数 Con0，Con1，Con2 しか許さない．

条件 [2] Z^0，Z^1 以外の変数の初期値は Con0，Con1，Con2 のいずれかとする．

条件 [3] 最初（0 番目）の命令は ROT に限定する．

条件 [4] ROT は連続して出現しない．

3.2 コーディングに使用するブール変数

3.1 節の疑似命令列存在問題を等価な SAT 問題にコーディングする上での工夫を述べる．二引数三値関数に対する疑似命令列存在問題では，trit 演算 f の引数となりうる 9 通りの組み合わせについて演算結果を確かめる必要がある．そこで，Z^0 と Z^1 の初期値を $Z^0 = 012012012t$，$Z^1 = 000111222t$ と設定することで全ての組み合わせを考慮することができる．このように初期値を設定し，疑似命令列実行後の A レジスタの値が $f(0,0)f(1,0)f(2,0)f(0,1)f(1,1)f(2,1)f(0,2)f(1,2)f(2,2)t$ になっていれば，$F(Z^0, Z^1)$ が実現できたといえる [1]．$f(0,0)\cdots f(2,2)t$ を **trit 演算 f の三進数九桁表現**と呼ぶ．例えば表 2 ならば $100102221t$ と表現できる．

以上で述べたコーディングを行うため，以下のブール変数を導入する．なお，以下では疑似命令を単に命令と書く．

- n 番目の命令を表すブール変数 $g_{n,i}$ （$0 \leq n <$ LENG，$0 \leq i \leq 4 +$ VNUM）

$$g_{n,i} = \begin{cases} \text{true} & \text{if } n \text{ 番目の命令が} \delta(i), \\ \text{false} & \text{o.w.} \end{cases}$$

ここで δ は次のように定める．$\delta(0)$ = ROT Con0，$\delta(1)$ = ROT Con1，$\delta(2)$ = ROT Con2，$\delta(3+k)$ = OPR Z^k ($0 \leq k < 2 +$ VNUM)．

- n 番目の命令実行後の A レジスタの d 桁目の値が t であることを表すブール変数 $a_{n,d,t}$ （$-1 \leq n <$ LENG，$0 \leq d \leq 8$，$0 \leq t \leq 2$）

$$a_{n,d,t} = \begin{cases} \text{true} & \text{if } n \text{ 番目の命令実行後の A レジスタの} \\ & d \text{ 桁目の値が } t, \\ \text{false} & \text{o.w.} \end{cases}$$

$n = -1$ に対する変数は 0 番目の命令実行前の A レジスタの初期値を表す．0 番目の命令を ROT に限定するため，$n = -1$ に対する変数はなくてもコーディングは可能であるが，制約の記

述を容易にするため用意している．

- n 番目の命令実行後の変数 Z^v の d 桁目の値が t であることを表すブール変数 $z_{v,n,d,t}$ $(0 \leqq v < 2 + \mathrm{VNUM}, -1 \leqq n < \mathrm{LENG}, 0 \leqq d \leqq 8, 0 \leqq t \leqq 2)$
- f の三進数九桁表現の d 桁目の値が t であることを表すブール変数 $tab_{d,t}$ $(0 \leqq d \leqq 8, 0 \leqq t \leqq 2)$

3.3 問題のコーディング

二引数三値関数に対する疑似命令列存在問題は以下で述べる論理式の積 φ_f で表せる．

制約 es1_g： 命令は一度に一つしか実行できないことを表す．

$$\bigwedge_{n=0}^{\mathrm{LENG}-1} \{g_{n,i} \mid 0 \leqq i < 4 + \mathrm{VNUM}\}_{\mathrm{ES}_1}$$

制約 es1_az： A レジスタと変数の値が well-defined であることを表す．

$$\bigwedge_{n=0}^{\mathrm{LENG}-1} \bigwedge_{d=0}^{8} \{a_{n,d,t} \mid 0 \leqq t \leqq 2\}_{\mathrm{ES}_1}$$

$$\bigwedge_{v=0}^{1+\mathrm{VNUM}} \bigwedge_{n=0}^{\mathrm{LENG}-1} \bigwedge_{d=0}^{8} \{z_{v,n,d,t} \mid 0 \leqq t \leqq 2\}_{\mathrm{ES}_1}$$

制約 initial_value： 各変数の初期値を表す．$\pi_v(d)$ は変数 Z^v の初期値の d 桁目の値を返す関数である．

$$\bigwedge_{d=0}^{8} z_{0,0,d,\pi_0(d)} \wedge \bigwedge_{d=0}^{8} z_{1,0,d,\pi_1(d)} \wedge$$

$$\bigwedge_{v=2}^{1+\mathrm{VNUM}} \Big(\bigwedge_{d=0}^{8} z_{v,0,d,0} \vee \bigwedge_{d=0}^{8} z_{v,0,d,1} \vee \bigwedge_{d=0}^{8} z_{v,0,d,2} \Big)$$

制約 check： LENG − 1 番目以下の命令実行後の A レジスタの値が入力の f の三進数九桁表現と等しいことを表す．

$$\bigvee_{n=0}^{\mathrm{LENG}-1} \bigwedge_{d=0}^{8} \bigwedge_{t=0}^{2} a_{n,d,t} \Longleftrightarrow tab_{d,t}$$

制約 exe_rot： ROT 命令の実行を表す．

$$\bigwedge_{i=0}^{2} \bigwedge_{n=0}^{\mathrm{LENG}-1} \Big(g_{n,i} \Longrightarrow \bigwedge_{d=0}^{8} a_{n,d,i} \wedge \bigwedge_{v=0}^{1+\mathrm{VNUM}} \bigwedge_{t=0}^{2} z_{v,(n-1),d,t} \Longleftrightarrow z_{v,n,d,t} \Big)$$

制約 exe_opr： OPR 命令の実行を表す．

$$\bigwedge_{v=0}^{1+\mathrm{VNUM}} \bigwedge_{n=0}^{\mathrm{LENG}-1} \Big(g_{n,(3+v)} \Longrightarrow \bigwedge_{\substack{0 \leqq t,t' \leqq 2 \\ \mathrm{op}(t,t')=t''}} \bigwedge_{d=0}^{8} \Big(a_{(n-1),d,t} \wedge z_{v,(n-1),d,t'} \Longrightarrow a_{n,d,t''} \wedge z_{v,n,d,t''} \Big) \wedge \bigwedge_{\substack{0 \leqq i' \leqq (1+\mathrm{VNUM}) \\ \wedge i' \neq i}} \bigwedge_{d=0}^{8} \bigwedge_{t=0}^{2} z_{i',(n-1),d,t} \Longleftrightarrow z_{i',n,d,t} \Big)$$

制約 firsti_is_rot： 最初の命令は ROT に限定することを表す．

$$g_{0,0} \vee g_{0,1} \vee g_{0,2}$$

制約 no_rot_sqe： ROT は連続して出現しないことを表す．

$$\bigwedge_{n=0}^{\mathrm{LENG}-2} \bigwedge_{i=0}^{2} (g_{n,i} \Longrightarrow \bigwedge_{i'=0}^{2} \neg g_{(n+1),i'})$$

4. 実　　装

ここでは本稿で提案する探索プログラムについて述べる．ここで**探索プログラム**とは疑似命令列存在問題を解くプログラムである．本稿の探索プログラムのアルゴリズムは以下の通りである．

Step1. f，LENG，VNUM を受け取る．

Step2. 受け取った入力から論理式 φ_f を作成する．

Step3. Step2 で作成した論理式を SAT ソルバに与える．

Step4. ソルバが SAT を返したならば，その割り当てから生成した疑似命令列と Success を返して終了．UNSAT ならば False を返す．

Step2 で作成される論理式が CNF のみのものと ES_1 節を許した CNF のものの二つのバージョンがある．

5. 実験と評価

文献 [1] の探索プログラムと本稿の探索プログラムの比較実験を行う．文献 [1] の探索プログラムを **PROG-EXIST** と呼ぶ．本稿の探索プログラムはアルゴリズムの Step2 で生成される論理式の形式，使用するソルバによって以下のように呼び分ける．

PROG-MINI： CNF のみの論理式を Minisat2 (ver 2-070721 simp) [4] で解く．

PROG-NANO： ES_1 節を許した CNF 論理式を Nanosat2.02 [5] で解く．

PROG-ES1SAT： ES_1 節を許した CNF 論理式を Es1sat [6] で解く．

実験手法は以下の通りである．

(i) 二引数三値関数の trit 演算をランダムに 100 個作成する．

(ii) その 100 個の trit 演算に対して，それぞれ VNUM を 2 に固定し，LENG = 5 と LENG = 10，LENG = 15 の三つの条件の下で測定する．一問のタイムアウトは 1200 秒とする．

実験に用いた計算環境は（CPU Xeon W5590（3.33GHz/4core 8thread/L2cache 4*256KB/L3cache 8MB）デュアル，メモリ 48GB）である．

実験結果を順に示す．実験結果は表とグラフにまとめた．表には各プログラムの出力と 100 問解くのにかかった時間を記し，グラフは横軸の時間内で解けた問題数を表している．表の total time にはタイムアウトの 1200 秒も含まれている．なお本稿の探索プログラムで問題のコーディングにかかる時間は非常に小さく，ソルバを走らせる時間が支配的であるため，実験データにコーディングにかかった時間を含めていない．

LENG = 5 **の場合**の結果を表 4 と図 1 にまとめた．本稿の探

表 4　LENG = 5, VNUM = 2
Table 4　LENG = 5, VNUM = 2

	Success	False	T.O.	total time(sec)
PROG-EXIST	2	98	0	1.585
PROG-MINI	2	98	0	4.124
PROG-NANO	2	98	0	7.866
PROG-ES1SAT	2	98	0	3.235

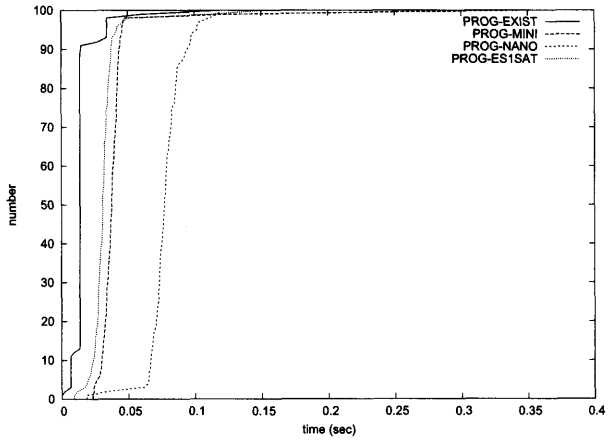

図 1　LENG = 5, VNUM = 2
Fig. 1　LENG = 5, VNUM = 2

表 5　LENG = 10, VNUM = 2
Table 5　LENG = 10, VNUM = 2

	Success	False	T.O.	total time(sec)
PROG-EXIST	55	45	0	33149
PROG-MINI	55	45	0	2609
PROG-NANO	54	7	39	67876
PROG-ES1SAT	55	45	0	3189

索プログラムで生成される論理式は，CNF のみの場合で変数数 721・節数 8430，ES_1 節を許した CNF の場合で変数数 721・節数 7650 であった．ほとんどの入力に対して False が返ってきており，このとき PROG-EXIST ではほぼ毎回全探索を行う必要があるのだが，このように探索範囲が非常に狭い条件では PROG-EXIST が最も早く 100 問の探索を終えている．

LENG = 10 **の場合**の結果を表 5 と図 2 にまとめた．本稿の探索プログラムで生成される論理式は，CNF のみの場合で変数数 1436・節数 19385，ES_1 節を許した CNF の場合で変数数 1436・節数 17825 であった．この条件では PROG-MINI と PROG-ES1SAT の二つで PROG-EXIST より高速に問題を解けている．しかし，PROG-NANO ではタイムアウトがかなり多くなっており，本稿の手法がソルバの性能に依存することが分かる．PROG-MINI などの結果から，PROG-NANO でタイムアウトになる問題の多くが False となる問題であることが分かる．

この条件では Success，False が約半分に分かれている．そこで Success になった問題，False なった問題についてまとめたグラフを図 3 と図 4 に示す．この二つのグラフから False になる問題がより多くの探索時間を必要とすることが分かる．特に

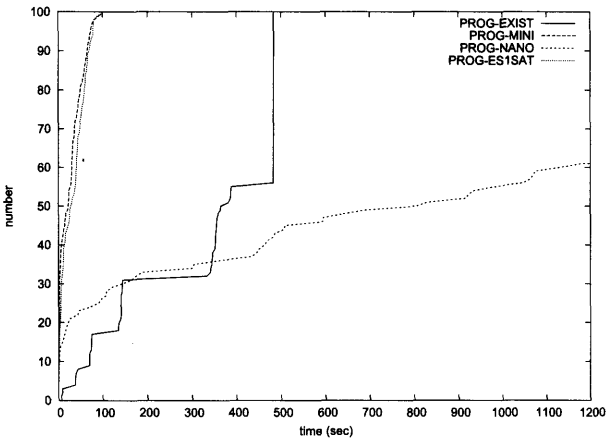

図 2　LENG =10, VNUM = 2
Fig. 2　LENG =10, VNUM = 2

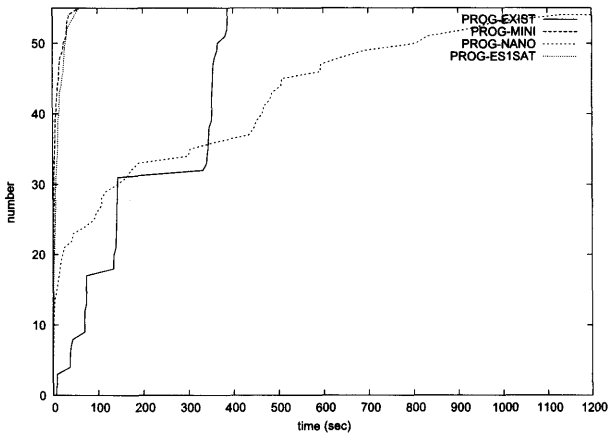

図 3　LENG = 10, VNUM = 2 Success
Fig. 3　LENG = 10, VNUM = 2 Success

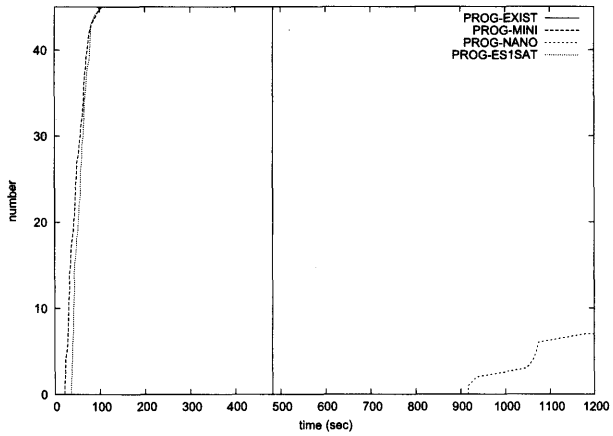

図 4　LENG = 10, VNUM = 2 False
Fig. 4　LENG = 10, VNUM = 2 False

PROG-NANO では False を出力するのに短くても 900 秒以上を要している．また PROG-EXIST は探索範囲が入力の表に依存しないため，False の時はこのように解くのにかかる時間が一定になっている．

LENG = 15 **の場合**の結果を表 6 と図 5 にまとめた．本稿の探索プログラムで生成される論理式は，CNF のみの場合

表 6　LENG = 15，VNUM = 2
Table 6　LENG = 15，VNUM = 2

	Success	False	T.O.	total time(sec)
PROG-EXIST	0	0	100	120000
PROG-MINI	99	0	1	10435
PROG-NANO	47	0	53	74689
PROG-ES1SAT	100	0	0	5795

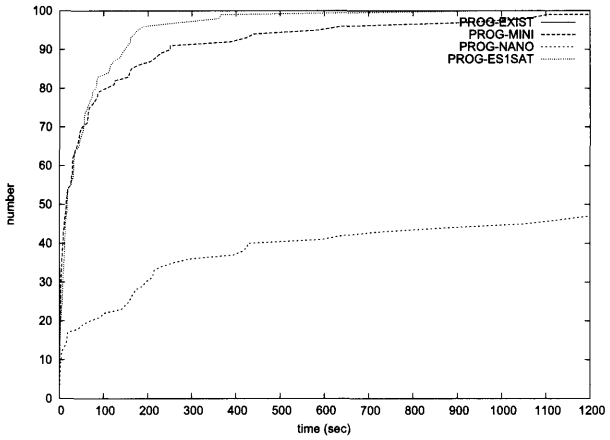

図 5　LENG =15，VNUM = 2
Fig. 5　LENG =15，VNUM = 2

で変数数 2151・節数 30340，ES_1 節を許した CNF の場合で変数数 2151・節数 28000 であった．LENG =10 の場合では PROG-MINI が最も良い結果を出しているが，この条件では 50 秒 ~60 秒の間に PROG-MINI を PROG-ES1SAT が抜かし，total time では PROG-ES1SAT が圧倒的に良い結果を出している．PROG-EXIST は全てでタイムアウトになっており，文献 [1] の手法の問題点が分かる．

PROG-ES1SAT の結果より，この条件では全ての問題が Success となる．つまり入力以外で変数を 2 個使えば，多くの表を長さ 15 以下の疑似命令列で実現可能だということが明かになった．

6. ま と め

低級アセンブリプログラミングの効率化のために，二引数三値関数の疑似命令列存在問題の SAT エンコーディングを提案した．実験により，本稿で提案したエンコーディングを利用した探索プログラムが既存手法より高速に，かつ，長い疑似命令列を発見できることを示した．本稿では二引数三値関数を扱ったが，本稿で提案した手法は n 引数三値関数に自然に拡張できる．

今後の課題としては，疑似命令列を低級アセンブリプログラムに変換する手法の開発が挙げられる．現在，疑似命令列から低級アセンブリプログラムへの変換は手作業で行われており，疑似命令列が長く複雑になるほどその変換には大変な労力が必要となる．そこで，使用できるフラグ数・プログラム中に記述できるフラグ数といった条件の下で入力の疑似命令列が低級アセンブリプログラムに変換できるか，という問題を SAT コーディングし，求められた割り当てから低級アセンブリプログラムを生成する手法を現在検討中である．この手法が開発できれば，入力ごとに異なる低級アセンブリプログラムを生成することができ，文献 [7] で提案された高級アセンブリプログラムからの Malbolge プログラム生成手法におけるインタプリタという固定の構造をなくすことができるため，文献 [1], [2], [9], [11] で指摘されている文献 [7] の手法の脆弱性を改善できると考えている．

謝辞　本研究は一部，科研費 #22650003 の助成を受けている．

文　献

[1] 安藤聡, 酒井正彦, 坂部俊樹, 草刈圭一朗, 西田直樹, Malbolge の高級アセンブリ言語への加算命令の追加，日本ソフトウェア科学会第 28 回大会講演論文集，No. 5A–3，12 pages，那覇，September 2011.
[2] 安藤聡, 酒井正彦, 坂部俊樹, 草刈圭一朗, 西田直樹, Malbolge の高級アセンブリ言語への配列機能の追加，電子情報通信学会ソフトウェアサイエンス研究会，電子情報通信学会技術報告 SS2012-8, Vol. 112, No. 23, pp. 43-49, 松山市, May 2012.
[3] Ben Olmstead, "Malbolge: Programming from Hell", http://www.bouletfermat.com/danny/malbolge/, 1998.
[4] Eén, N. and Sorensson, N. An extensible SAT-solver. Lecture notes in computer science, Vol. 2919, pp. 502–518, 2004.
[5] 日野善信, 酒井正彦, 草刈圭一朗, 坂部俊樹, 西田直樹, 2 カウンタ法に基づく基本対称節を持つ CNF 論理式の SAT ソルバ，平成 22 年度電気関係学会東海支部連合大会講演論文集，春日井市，No. D3-5, 1 page, August 2010.
[6] 日野善信, 酒井正彦, 坂部俊樹, 草刈圭一朗, 西田直樹, 2 リテラル監視法で実装された SAT ソルバへの基本対称節処理機能の組み込み，電子情報通信学会ソフトウェアサイエンス研究会，電子情報通信学会技術研究報告 SS2011-38, Vol. 111, No. 268, pp. 67-72, 能美市, October 2011.
[7] 飯澤恒，難解言語 Malbolge に基づくプログラム難読化に関する研究，名古屋大学修士論文，2006.
[8] 飯澤恒，坂部俊樹，酒井正彦，草刈圭一朗，西田直樹，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，信学技報，電子情報通信学会，Vol.105，No. 129，pp. 25-30，2005.
[9] 菅優也，難読言語 Malbolge の逆コンパイル困難性に関する研究，高知工科大学卒業論文，2011.
[10] Lou Scheffer, Introduction to Malbolge, http://www.lscheffer.com/malbolge.html.
[11] 長坂哲，酒井正彦，坂部俊樹，草刈圭一朗，西田直樹，難解言語 Malbolge のチューリング完全性について，信学技報，電子情報通信学会，Vol. 110，No. 227，pp. 55–60，2010.
[12] Malbolge, Wikipedia, http://en.wikipedia.org/wiki/Malbolge.