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
SS2012-50(2013-01)

Malbolge低級アセンブリプログラミングにおける制御命令の配置設計のためのSATソルバの利用

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，低級アセンブリプログラムでは変数を引数とする命令はその変数宣言の直前に記述しなければならず実行制御が必要不可欠なことと，制御命令には無条件ジャンプとアクセスの度にジャンプとスルーが入れ替わるフリップフロップジャンプしか存在しないことから，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．これまで制御命令の配置設計は手作業により行われており，実行トレーサを利用しているものの，非常に困難であった．本稿では，Malbolgeの低級アセンブリプログラミングにおける制御命令の配置設計に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，制御命令の配置問題を定式化し，その問題のSATエンコーディングを提案する．実験により，提案手法を利用した制御命令配置設計ツールの性能を評価する．

キーワード 難解プログラミング言語，Malbolge，SATソルバ，制御命令の配置設計

Using SAT Solvers for Solving Control-Instruction Layout Problems in Low-Level Assembly Programming for Malbolge

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 as one of the most esoteric programming languages. Although it became possible to write programs in Malbolge due to the development of the low-level assembly language, we still have a problem that the programming in the low-level assembly language is difficult. This is because for each variable, instructions that take the variable in their arguments are allowed to be placed just above of the variable in the low-level assembly programs, and control-instructions in the low-level assembly language are only unconditional jumps and flip-flop jumps, where the latter alternate jump and no-operations. So far we have to design control structures, i.e., layout of control-instructions, by hand, which is very difficult even if we have an execution tracer for low-level assembly programs. In this paper, to solve this problem, we propose a method to design layout of control-instructions of the low-level assembly language efficiently by using state-of-art SAT solvers. We define a control-instruction layout problem, and propose a SAT encoding for this problem. We also evaluate the performance of control-instruction layout tools based on our method.

Key words Esoteric programming language, Malbolge, SAT solver, Control-Instruction Layout

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 [5], [6]，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依然として容易ではない．

これまでのプログラム開発手法は，まず低級アセンブリ言語の命令系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しており [1], [5]〜[7]，この手法における疑似命令での設計を支援するツールも構築されている [2]．低級アセンブリプログラムでは変数を引数とする命令はその変数宣言の直前に記述しなければならないという制約があり，疑似命令列からのコーディングでは同じ命令を複数回実行するための実行制御が必要不可欠である．しかし，低級アセンブリ言語の制御命令は無条件ジャンプかアクセスの度にジャンプとスルーが入れ替わるフリップフロップジャンプ（FF ジャンプ）しか存在せず，FF ジャンプの配置設計が実行制御の大部分を占める．これまでこの配置設計は手作業で行われており，低級アセンブリプログラムの実行トレーサ [8] を利用しているものの，非常に困難であった．

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 ソルバの性能が飛躍的に向上したことを受け，SAT 問題への変換による問題解法が進んでいる．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，FF ジャンプの配置設計に 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し，疑似命令列から低級アセンブリプログラムへの変換ツールを実装する．実験により，このツールの性能を評価する．

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 FF ジャンプの配置問題を定式化し，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変換ツールの実装を述べる．5. 節では実装したツールの性能評価実験を行い，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 [5], [6], [12] を参照されたい．

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．`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．以下に関数 OP と，`Rot` で用いられる関数 ROTR の計算例を示す．

OP(0120120120t, 0001112222t) = 1001022212t

ROTR(0001112222t) = 2000111222t

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

2.2 低級アセンブリ言語

低級アセンブリ言語 [5] 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．PC はプログラムカウンタである．低級アセンブリプログラムは低級アセンブラ [5] を用いて Malbolge プログラムに変換できる．

低級アセンブリ言語の命令を表 3 にまとめた．[X] は X 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．`JMP_UNIT` と `MOV_PC_UNIT`，`FF`i，`FLIP`，そしてその復元命令を**制御命令**，それ以外を非制御命令と呼ぶ．以降で，単に命令と書いた場合は非制御命令を指す．`FF`i は**フリップフロップジャンプ**（FF ジャンプ）と呼ばれ，アクセスの度にジャンプとスルーが入れ替わる．`FF`i の i を FF ジャンプの**添字**と呼び，添字によって複数の FF ジャンプを管理する．添字が同一の FF ジャンプらは，異なる位置に置かれていたとしても，プログラムの実行中に常に ON，OFF が一致している．どの FF ジャンプも初期状態は ON である．

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複数付加可能な命令の列と変数定義から構成される．図 1 にプログラム例を示す．":" の左側がラベルである．"ENTRY" ラベルがエントリーポイントを表す．図 1 のプログラムでは，最初の `FF0` へのアクセスではラベル `OPR_Y` にジャンプするが二度目はジャンプしない．これにより図 1 の右のように命令が実行される．プログラム中に記述された FF ジャンプの総数を

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 X	A,[X] := ROTR([X])
REV_ROT	ROT_UNIT 命令の復元
OPR_UNIT X	A,[X] := OP(A,[X])
REV_OPR	OPR_UNIT 命令の復元
INPUT_UNIT	A:=getchr()
REV_INPUT	INPUT_UNIT 命令の復元
OUTPUT_UNIT	putchr(A)
REV_OUTPUT	OUTPUT_UNIT 命令の復元
JMP_UNIT *label*	PC:= *label*
MOV_PC_UNIT *label*	PC:= *label*
REV_MOV_PC	MOV_PC_UNIT 命令の復元
FFi	FFi が ON 場合 : PC:= *label*
label	FFi が OFF の場合 : 何もしない
	アクセスの度に ON, OFF が入れ替わる
FLIP FFi	FFi がフリップされる (ON↔OFF)
END	終了

```
ENTRY:
ROT_CON2:   ROT_UNIT CON2
CON2:       2222222222t
            REV_ROT
OPR_X:      OPR_UNIT X
X:          0000000001t
            REV_OPR
OPR_Y:      OPR_UNIT Y
Y:          2222222222t
            REV_OPR
            FF0
               OPR_Y
            END
```

実行系列
ROT_UNIT CON2
REV_ROT
OPR_UNIT X
REV_OPR
OPR_UNIT Y
REV_OPR
OPR_UNIT Y
REV_OPR
END

図 1 低級アセンブリプログラムの例と実行される命令系列
Fig. 1 Example of Low-Level Assembly Program

FF ジャンプの**配置数**と呼び，その添字のバリエーションを FF ジャンプの**種類数**と呼ぶ．図 1 のプログラムの FF ジャンプの配置数は 1，種類数も 1 である．

図 1 のような実行系列において復元命令は本質的ではなく，以下で述べるプログラミング上の制限からもたらされたものである．そこでプログラムの設計段階では復元命令を除いたものを与えることにして，これを疑似命令 [5] と呼ぶ．図 1 の実行系列は疑似命令列を用いて図 2 のように与えられる．

低級アセンブリプログラミングには次の制限が伴う．

制限 1. どの変数も一箇所にしか定義できない．

制限 2. 変数を引数とする命令は，その変数定義の直前に記述しなければならない．

制限 3. JMP_UNIT と END，FFi 以外の命令実行後は，次にその命令を実行する前にその復元命令を実行するように記述しなければならない．

制限 1,2 より，同一変数を引数とする命令を繰り返し実行する場合には実行制御が必要となる．低級アセンブリ言語には FF ジャンプと無条件ジャンプしかないため，そのプログラミングでは FF ジャンプの配置設計が鍵となる．また制限 3 を満たすように復元命令を適切に記述することも重要である．

```
ROT  CON2
OPR  X
OPR  Y
OPR  Y
END
```

図 2 疑似命令列の例
Fig. 2 Example of Pseudo-Instruction Sequence

2.3 SAT 問題

SAT 問題とは，ブール変数とその否定をリテラルとする CNF 論理式が与えられたとき，その式が充足可能（SAT）かどうかの決定問題である．SAT 問題を解くツールは SAT ソルバと呼ばれ，式が SAT ならば証拠となる変数への値割り当てを返す．

ES_k 節とは節中のリテラルのうち，ちょうど k 個だけが 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 [11]．リテラル $l_1, \ldots, l_m$ からなる ES_k 節を $\{l_1, \ldots, l_m\}_{\mathrm{ES_k}}$ で表す．$k = 1$ の場合の ES_1 節は以下の論理式と等価である．

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

3. SAT エンコーディング

本節では疑似命令列から低級アセンブリプログラムへの変換手法を提案する．3.1 節ではこの変換における FF ジャンプの配置設計を決定問題として定式化する．3.2 節では，3.1 節で定義した問題を 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 節でコーディング法について述べる．

3.1 制御命令配置問題

低級アセンブリプログラミングにおける制御命令配置問題は，図 2 のような疑似命令列を実行系列とする図 1 のような低級アセンブリプログラムの存在を問うものであり，次のように定義される．

制御命令配置問題
インスタンス： END で終わる疑似命令列 QI，二つの自然数 FKIND と PNUM.
解： FKIND 種類以下の FF ジャンプを PNUM 個以下配置して，実行される命令系列が QI となる低級アセンブリプログラムが存在するか？

本稿では問題の解に次の条件を課す．

条件 1. 命令と引数，その復元命令はひとかたまりで扱い，この中に制御命令は配置しない．以降では命令と書くだけでこのコードブロックを指すものとする．例えば “変数 X に対する OPR_UNIT 命令” とは次のコードブロックを指し，これは疑似命令 “OPR X” と対応する．

```
    OPR_UNIT X
X:  ...
    REV_OPR
```

条件 2. 各命令間に，添字の異なる 0 個以上の FF ジャンプを添字が昇順になるように配置する．

条件 3. エントリーポイントと最初の命令の間に FF ジャンプ

のフリップを配置してもよい.

条件 4. 条件 2,3 を満たさない制御命令は配置しない.

インスタンスの QI についても QI **中に現れるどの変数も** OPR **と** ROT **両方の引数になることはない**という条件を課す. これは既存研究 [1], [5]〜[7] において変数に OPR と ROT 両方を使う疑似命令列が少ないためである. 任意の QI を考慮した SAT エンコーディングについては紙面の都合上省略したので [13] を参照されたい.

3.2 コーディングに使用するブール変数

問題のコーディングで使用するブール変数を導入する. QI の命令列長と命令の種類をそれぞれ LENG と IKIND で表す. 例えば図 2 の LENG は 5, IKIND は 4 である. 以下では, ブール変数 x に対して「x : P」という記法を用い, P の状態であるとき, かつその時に限り x が true となることを表す.

$\boldsymbol{pos}_{p,i}$: 低級アセンブリプログラムで p 番目の命令が i である. i は QI で i 番目に出現する疑似命令と対応する命令とする. ($0 \leqq p <$ IKIND, $0 \leqq i <$ IKIND)

$\boldsymbol{pc}_{s,p}$: s ステップ目にプログラムカウンタが p 番目の命令を指している. ($0 \leqq s <$ LENG, $0 \leqq p <$ IKIND)

$\boldsymbol{exe}_{s,i}$: s ステップ目に命令 i を実行する. ($0 \leqq s <$ LENG, $0 \leqq i <$ IKIND)

$\boldsymbol{ff\_set}_{p,f}$: p 番目と $p+1$ 番目の命令の間に添字 f の FF ジャンプを置く. ($0 \leqq p <$ IKIND $- 1$, $0 \leqq f <$ FKIND)

$\boldsymbol{ff\_branch}_{p,f,p2}$: p 番目と $p+1$ 番目の命令の間の添字 f の FF ジャンプの分岐先が $p2$ 番目の命令である. ($0 \leqq p <$ IKIND $- 1$, $0 \leqq f <$ FKIND, $0 \leqq p2 <$ IKIND)

$\boldsymbol{ff\_cond}_{s,f}$: s ステップ目に添字 f の FF ジャンプの状態が ON である. ($0 \leqq s <$ LENG, $0 \leqq f <$ FKIND)

3.3 問題のコーディング

制御命令配置問題は以下で述べる制約を表す論理式の積 φ_{QI} で表せる. ページの都合上, 一部の論理式のみ示す. 全ての論理式は [13] を参照されたい. 以降で, **場所** p とは低級アセンブリプログラムの p 番目の命令の位置を指す.

制約 es1_pos1: どの場所にも命令は一つしか置けない.

$$\bigwedge_{p=0}^{\mathrm{IKIND}-1} \{pos_{p,i} \mid 0 \leqq i < \mathrm{IKIND}\}_{\mathrm{ES}_1} \tag{1}$$

制約 es1_pos2: どの命令も一つの場所にしか置けない.

制約 es1_pc: どのステップでもプログラムカウンタの指す位置は一つである.

制約 es1_exe: どのステップでも実行できる命令は一つである.

制約 es1_ff_branch: FF ジャンプを置くならば, その引数は一つである.

制約 check: 実行される命令の系列が QI である.

制約 execution: 各ステップでプログラムカウンタが指している場所に置かれた命令を実行する.

$$\bigwedge_{s=0}^{\mathrm{LENG}-1} \bigwedge_{p=0}^{\mathrm{IKIND}-1} \bigwedge_{i=0}^{\mathrm{IKIND}-1} pc_{s,p} \wedge pos_{p,i} \Longrightarrow exe_{s,i} \tag{2}$$

制約 control_pc: 各ステップで, プログラムカウンタの位置とその後ろに配置された FF ジャンプの状態から次のステップのプログラムカウンタの位置が定まる.

$$\bigwedge_{s=0}^{\mathrm{LENG}-2} \bigwedge_{p=0}^{\mathrm{IKIND}-2} pc_{s,p} \Longrightarrow \mathbf{hand}(s,p,0) \tag{3}$$

ここで $\mathbf{hand}(s,p,f)$ という略記法は次の論理式を表している. 場所 p と場所 $p+1$ の間に添字 f の FF ジャンプが置いてあり s ステップ目で ON ならば, $s+1$ ステップ目のプログラムカウンタが指す場所は添字 f の FF ジャンプの引数のラベルのついた場所である. その他の場合は $\mathbf{hand}(s,p,f+1)$ となる.

- $0 \leqq f \leqq$ FKIND $- 2$ の時

$$\begin{aligned}
&\Big(\bigwedge_{p2=0}^{\mathrm{IKIND}-1} ff_set_{p,f} \wedge ff_cond_{s,f} \wedge ff_branch_{p,f,p2} \Longrightarrow \\
&pc_{(s+1),p2} \wedge \bigwedge_{f2=f+1}^{\mathrm{FKIND}-1} \big(ff_cond_{s,f2} \Longleftrightarrow ff_cond_{(s+1),f2}\big) \\
&\qquad \wedge \mathbf{ff_cond_cons}(s,p,f)\Big) \wedge \\
&\big(ff_set_{p,f} \wedge \neg ff_cond_{s,f} \Longrightarrow \mathbf{hand}(s,p,f+1)\big) \wedge \\
&\big(\neg ff_set_{p,f} \Longrightarrow \mathbf{hand}(s,p,f+1)\big)
\end{aligned} \tag{4}$$

- $f =$ FKIND $- 1$ の時

$$\begin{aligned}
&\Big(\bigwedge_{p2=0}^{\mathrm{IKIND}-1} ff_set_{p,f} \wedge ff_cond_{s,f} \wedge ff_branch_{p,f,p2} \Longrightarrow \\
&\qquad pc_{(s+1),p2} \wedge \mathbf{ff_cond_cons}(s,p,f)\Big) \wedge \\
&(ff_set_{p,f} \wedge \neg ff_cond_{s,f} \Longrightarrow \\
&\qquad pc_{(s+1),(p+1)} \wedge \mathbf{ff_cond_cons}(s,p,f)) \wedge \\
&\big(\neg ff_set_{p,f} \Longrightarrow pc_{(s+1),(p+1)} \wedge \mathbf{ff_cond_cons}(s,p,f)\big)
\end{aligned} \tag{5}$$

ここで $\mathbf{ff\_cond\_cons}(s,p,f)$ という略記法は次の論理式を表している. 添字が f 以下の FF ジャンプについて, 場所 p と場所 $p+1$ の間に置いてあるならば s ステップ目と $s+1$ ステップ目で ON/OFF が入れ替わり, 置いてないならば s ステップ目と $s+1$ ステップ目で ON/OFF は変わらない.

$$\begin{aligned}
&\bigwedge_{f2=0}^{f} \Big(\big(ff_set_{p,f2} \Longrightarrow (ff_cond_{s,f2} \Longleftrightarrow \neg ff_cond_{(s+1),f2})\big) \wedge \\
&\big(\neg ff_set_{p,f2} \Longrightarrow (ff_cond_{s,f2} \Longleftrightarrow ff_cond_{(s+1),f2})\big)\Big)
\end{aligned} \tag{6}$$

制約 setff_num: FF ジャンプが PNUM 個以下配置される.

$$\bigvee_{k=0}^{\mathrm{PNUM}} \{ff\_set_{p,f} \mid 0 \leqq p < \mathrm{IKIND}-1, 0 \leqq f < \mathrm{FKIND}\}_{\mathrm{ES}_k} \tag{7}$$

制約 starting_pc: プログラムカウンタの初期値が位置 0.

制約 end_prog: END はプログラムの最後にある.

$$pos_{(\mathrm{IKIND}-1),(\mathrm{IKIND}-1)} \tag{8}$$

4. 実　　装

低級アセンブリプログラミングでは，メモリの制限から，プログラム内の FF ジャンプの種類数と配置数をできるだけ少なくする配置設計が求められる．これは最適化問題として捉えることができる．そこで本節では，まず 4.1 節で決定問題である制御命令配置問題を解くツールを実装し，そして 4.2 節でそのツールを繰り返し用いることで FF ジャンプの配置設計に関する最適化問題を解くツールを実装する．

4.1 決定問題を解くツール

3. 節で提案した SAT エンコーディングを基に，制御命令配置問題を解くツール LAYOUT–BY–SAT（LBS）を実装する．

LAYOUT–BY–SAT
入力: QI，FKIND，PNUM.
出力: Success と低級アセンブリプログラム，あるいは Fail.
アルゴリズム
Step1: 入力から論理式 φ_{QI} を生成する．
Step2: φ_{QI} を SAT ソルバに与える．
Step3: ソルバが SAT を返したならば，その値割り当てから定まる低級アセンブリプログラムと Success を返して終了．UNSAT ならば Fail を返して終了．

Step1 で生成される論理式の形式は，Step2 で使用するソルバに応じて，CNF のみか ES_1 節を許した CNF に切り替える．3.3 節で示した論理式の中で ES_1 節を含まないものについては，制約 execution は分配則，制約 control_pc は Tseitin 変換 [10]，制約 setff_num は Totalized [9] を用いて CNF に変換する．

4.2 最適化問題を解くツール

最適化問題は決定問題を繰り返し解くことによってその解を得られることが知られている．4.1 節で実装したツールを繰り返し利用することで，FF ジャンプの種類数，あるいは配置数を最小にする配置設計に基づいた低級アセンブリプログラムを求めるツールを実装する．

MIN–FKIND
入力: QI.
出力: 実行系列が QI で，かつ FF ジャンプの**種類数を最小**にする配置設計に基づいた低級アセンブリプログラム．
アルゴリズム
Step1: 変数 FK，PN にそれぞれ 1 を代入．
Step2: QI，FK，PN を入力として LBS を実行．
Step3: LBS が Success を返したならば，その低級アセンブリプログラムを出力して終了．さもなくば Step4 へ．
Step4: PN が $(\mathrm{IKIND}-1)*FK$ より小さいならば，PN をインクリメントして Step2 へ戻る．さもなくば Step5 へ．
Step5: FK をインクリメントし，PN にそれを代入して Step2 へ戻る．

MIN–PNUM
入力: QI.
出力: 実行系列が QI で，かつ FF ジャンプの**配置数を最小**にする配置設計に基づいた低級アセンブリプログラム．
アルゴリズム
Step1: 変数 FK，PN にそれぞれ 1 を代入．
Step2: QI，FK，PN を入力として LBS を実行．
Step3: LBS が Success を返したならば，その低級アセンブリプログラムを出力して終了．さもなくば Step4 へ．
Step4: FK が PN より小さいならば，FK をインクリメントして Step2 へ戻る．さもなくば Step5 へ．
Step5: PN をインクリメントし，$PN \leqq (\mathrm{IKIND}-1)*n$ を満たす最小の n を FK に代入して Step2 へ戻る．

5. ツールの性能評価実験

4.2 節で実装したツールの性能評価実験を行う．実験は次の三つの項目について，各ツール毎に行う．

1. 手変換との比較
2. 実時間以内に変換できる QI の長さの最大値
3. QI 中の命令の種類の多寡が変換に与える影響

問題のサイズは疑似命令列の命令列長と命令の種類の組で与え（LENG，IKIND）と表す．実験は全て計算環境（CPU Xeon W5590 （3.33GHz/4core 8thread/L2cache 4*256KB/L3cache 8MB）デュアル，メモリ 48GB）で行う．LBS が Step2 で用いるソルバは Minisat2 (ver 2-070721 simp) [4] とし，それに合わせて Step1 で生成される論理式は CNF のみの形式とする．

5.1 実験 1：手変換との比較

実験 1 では手作業で行った変換と 4.2 節で実装した各ツールによる変換で，どちらが FF ジャンプの配置数・種類数を少なく抑えて変換できるかを調査する．実験は既存研究 [1], [6] で低級アセンブリプログラムにコーディングされた疑似命令列 copy，carry，sum の三つについて行う．各問題のサイズは，copy が（12，6），carry が（10，6），sum が（18，8）である．

実験結果を表 4 にまとめた．表は各変換方法で問題の疑似命令列のコーディングに用いた FF ジャンプの配置数・種類数を表す．本稿のツールは変換に要した時間も記した．問題のサイズが小さい copy や carry では手変換の方が良い結果となっている．これは本稿で解に与えた条件が厳しすぎることを示しており，同時に条件を弱めればもっと少ない配置数・種類数での変換が可能であることを示している．問題のサイズが大きい sum では手での配置設計が困難であるため，本稿のツールらが良い結果を出している．

5.2 実験 2：変換できる QI の長さの最大値

実験 2 では各ツールで変換できる QI の命令列長の最大値を調査する．実験は各サイズ 10 個の疑似命令列をランダムに用意しツールに与え，10 問中 5 問以上タイムアウトになるまでサイズを増やして実験を続ける．一問のタイムアウトは 86400

表 4　手変換とツールとの比較
Table 4　Comparison between hand and tool

変換方法	項目	copy	carry	sum
手変換	FKIND	2	2	7
	PNUM	5	3	11
	time(sec)	–	–	–
MIN–FKIND	FKIND	3	2	4
	PNUM	6	5	11
	time(sec)	0.95	0.41	18.86
MIN–PNUM	FKIND	3	2	4
	PNUM	6	5	11
	time(sec)	1.06	0.72	57.05

表 5　変換できる QI の長さの最大値
Table 5　The maximum of the length of QI that tools can convert

	MIN–FKIND		MIN–PNUM	
Size	Solved/T.O.	Median(sec)	Solved/T.O.	Median(sec)
(20,8)	10 / 0	177.41	10 / 0	1427.74
(21,8)	9 / 1	274.92	6 / 4	216.58
(22,8)	8 / 2	606.88	5 / 5	893.30
(23,8)	6 / 4	612.46	–	–
(24,8)	5 / 5	1309.50	–	–

表 6　命令の種類の変化が変換に与える影響
Table 6　The influence for conversion that the change of the kind of instruction in QI

	MIN–FKIND		MIN–PNUM	
Size	Solved/T.O.	Median(sec)	Solved/T.O.	Median(sec)
(20,4)	5 / 5	17.44	5 / 5	35.32
(20,8)	9 / 1	60.79	10 / 0	934.99
(20,16)	10 / 0	18.26	10 / 0	37.27

秒とする．

実験結果を表 5 にまとめた．表には各ツールでその行のサイズの問題のうち “解けた問題数/タイムアウトの問題数” と解けた問題に要した時間の中央値を記した．結果から，命令の種類が 8 ならば，MIN–PNUM は 21，MIN–FKIND は 23 の長さの疑似命令列まで変換できることが分かった．

5.3　実験 3：QI 中の命令の種類の多寡が変換に与える影響

実験 3 では QI 中の命令の種類の多寡がツールによる変換にかかる時間にどのように影響を与えるのかを調査する．実験は各サイズ 10 個の疑似命令列をランダムに用意しツールに与える．一問のタイムアウトは 86400 秒とする．

実験結果を表 6 にまとめた．表 6 より，命令の種類が少ない問題，つまり FF ジャンプでの実行制御をより多く必要とする問題の方が解くのに多くの時間を要することが分かる．ただし解けた問題の中央値から，命令の種類が 4 の場合でも短い時間で解ける問題があることが分かり，問題の難しさは解いてみなければ分からないという結論に至った．

6. ま と め

低級アセンブリプログラムの開発支援のために，制御命令配置問題の SAT エンコーディングを提案した．実験により，実装した疑似命令列から低級アセンブリプログラムへの変換ツールがどの程度の性能をもつのかを示した．

今後の課題としては，高級アセンブリプログラム [5], [6] から Malbolge プログラムを生成するコンパイラの開発が挙げられる．現在，この生成は低級アセンブリ言語で実装されたインタプリタの内部に高級アセンブリプログラムを埋め込む形で行われており，文献 [1], [7], [8] で固定の構造から生まれる脆弱性が指摘されている．そこで高級アセンブリプログラムを疑似命令列に変換し，それを低級アセンブリプログラムに変換することで，入力ごとに異なる構造のプログラムを生成する手法を検討している．しかし実験 2 より本稿の手法で変換できる疑似命令列の長さは 21 ～ 23 程度であり，検討手法には十分ではなく，より長い疑似命令列をどのように変換するかが課題となる．

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, 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, 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 SS2012–37, Vol. 112, No. 275, pp. 7–12, 広島市, November 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] 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.
[6] 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.
[7] 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.
[8] 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 の弱チューリング完全性とプログラミング環境, 名古屋大学修士論文, 2011.
[9] Oliver Bailleux, Yacine Boufkhad, and Olivier Roussel, New Encodings of Pseudo-Boolean Constraints into CNF, O.Kullmann(Ed.):SAT 2009, LNCS 5584, pp.181–194,2009.
[10] G.S. Tseitin, On the complexity of derivation in propositional calculus, Studies in constructive mathematics and mathematical logic, Vol. 2, No. 115–125, pp. 10–13, 1968.
[11] 馬野洋平, 酒井正彦, 西田直樹, 坂部俊樹, 草刈圭一朗, 基本対称関数を付加した CNF 論理式の充足可能性判定. 電子情報通信学会論文誌, Vol. J93–D, No. 1, pp.1–9, 2010.
[12] Malbolge, Wikipedia, http://en.wikipedia.org/wiki/Malbolge.
[13] http://www.trs.cm.is.nagoya-u.ac.jp/~ando/