アルゴリズム 7－3
（1989. 5. 22）

ビットの操作に関するアルゴリズム

中森 眞理雄

東京農工大学 工学部 数理情報工学科

あらまし

大きさがn×nのビットマトリクスを回転したり左右／上下／斜めに裏返したりする一群の算法を考察する．周知の通り，それらの算法は非可換群をなす．本論文では，それらの算法が簡単な書換え規則により互いに移り変わることを示す．さらに，これらの書換え規則が可換群あるいは非可換群をなすことを示し，算法の群との違いを述べる．

ON THE STRUCTURE OF THE OPERATOR GROUP ON ALGORITHMS OF BIT MANIPULATION

Mario Nakamori

Faculty of Technology
Tokyo University of Agriculture and Technology
Koganei-shi, 184 Japan

Abstract

Algorithms of rotating and/or transposing an n by n bit matrix are considered, where the (i, j)-th element of the matrix is located in the j-th bit of the i-th word. As is well known, such altorithms form a noncommutative group. In the present paper it is shown that there are simple rules of translating an algorithm to another and that these rules form a group. The structure of the group is also investigated.

1．まえがき

文字，記号などを表すｎ×ｎドットマトリクスが，１語がｎビットの計算機の連続するｎ語に格納されているとする．ただし，このマトリクスの第ｉ行第ｊ列が第ｉ語の第ｊビットに対応する（ｉ，ｊ＝０，１，…，ｎ－１）．このマトリクスを反時計回りに９０°，１８０°，２７０°回転する問題や，上下，左右，斜めの軸に関して裏返す問題を考える（表１）．以下では，これらの操作を“回転等”と総称する．例として，９０°回転の場合を図１に示す．周知のとおり，これらの回転等は群をなす（表２）．

素朴にｎ^{2}個のビットの各々を対応する場所に移動する算法はＯ（ｎ^2）の手間を要する．パターンを上下，左右に４分割することを繰り返して４分木（quad tree）として表現すれば，回転等はポインタのつけ換えでできる[1]が，データ構造の変換にＯ（ｎ^2）の手間を要する．

分割統治法を用いれば回転等はＯ（ｎlogｎ）の手間で可能である[2]，[3]．これらの算法は，後述の“ブロックを単位とする回転等”を基本的な操作とするもので，いくつかの命令を書き換えることにより相互に移り変わる．この書換えを算法に対する演算子と見なすことにすると，それらの演算子は上記の回転等そのものの群とは異なる群をなす．本論文は，この演算子群の構造を論ずる．

本研究は“浄書システム”の開発[4]における プリンタ部の設計に関して相談を受けたことに端を発している．そこでは，プリンタ部にモトローラ社のＭＣ６８０００を用い，３２×３２ドットマトリクスによる文字を１文字当り５ミリ秒以内で回転することが要求された．　素朴にｎ^{2}個のビットの各々を 対応する場所に移動する算法では約８ミリ秒かかる．分割統治法によるアルゴリズムでは約３ミリ秒で可能である．

表１　回転等の算法

I	R_{90}	R_{180}	R_{270}
			
V	G	H	D
			

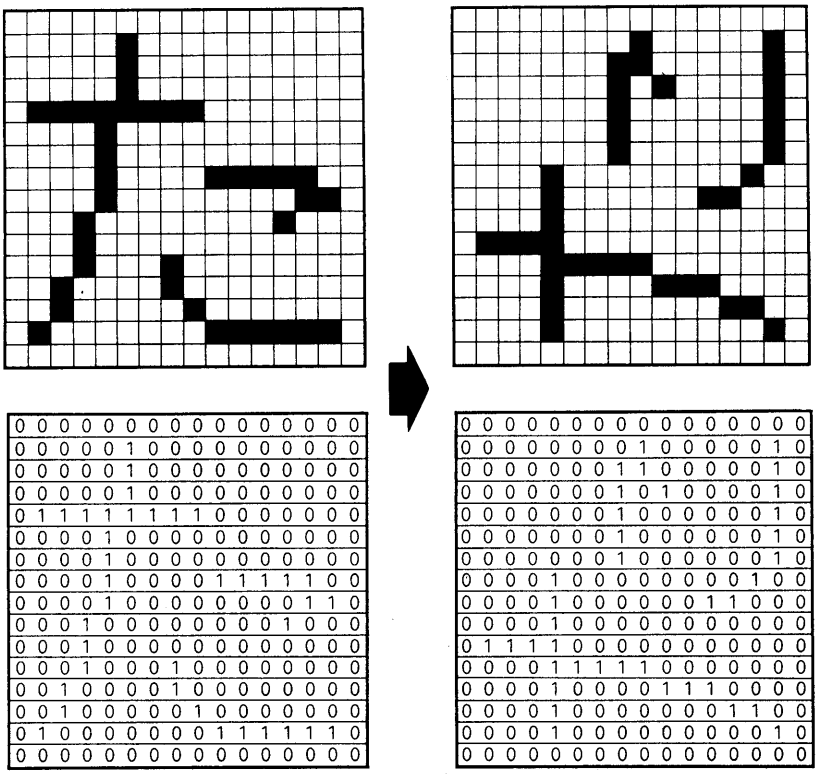


図１　ドットマトリクスの９０°回転

Fig.1　Rotation of a dot matrix by 90°

表２　回転等の算法の群表

算法Xに引き続いて算法Yを実行したときの結果

Y \ X	I	R_{90}	R_{180}	R_{270}	V	G	H	D
I	I	R_{90}	R_{180}	R_{270}	V	G	H	D
R_{90}	R_{90}	R_{180}	R_{270}	I	D	V	G	H
R_{180}	R_{180}	R_{270}	I	R_{90}	H	D	V	G
R_{270}	R_{270}	I	R_{90}	R_{180}	G	H	D	V
V	V	G	H	D	I	R_{90}	R_{180}	R_{270}
G	G	H	D	V	R_{270}	I	R_{90}	R_{180}
H	H	D	V	G	R_{180}	R_{270}	I	R_{90}
D	D	V	G	H	R_{90}	R_{180}	R_{270}	I

2では表１の諸算法の統一的な記述を示し，一部の定数を指定することにより各算法が特徴づけられることを示す．**3**では表１の回転等の諸算法を，一部を書き換えることにより，相互に変換する方法を述べる．**4**では，算法を書き換える演算子のなす群の構造を述べる．

2．回転等の諸算法の統一的な記述

以下では　n＝2^mとする（nが２のべきでない場合の算法については文献(2)を参照されたい）．

次の算法を考えよう．

```
blocksize:=2;
for k:=1 step 1 until m
 begin
  for u:=0 step blocksize until n-1 do
   for w:=0 step 1 until blocksize/2 do
    begin
     register0 :=
       matrix[source0[k]+u+w]∧mask0[k];
     register1 :=
       matrix[source1[k]+u+w]∧mask1[k];
     register2 :=
       matrix[mod(source1[k]+blocksize/2,
            blocksize)+u+w]∧¬mask0[k];
     register3 :=
       matrix[mod(source1[k]+blocksize/2,
            blocksize)+u+w]∧¬mask1[k];
     register0を左へshift0[k]ビットシフト;
     register1を右へshift1[k]ビットシフト;
     register2を右へshift0[k]ビットシフト;
     register3を左へshift1[k]ビットシフト;
     matrix[destination0+u+w] :=
       register0∨register1;
     matrix[destination1+u+w] :=
       register2∨register3;
    end
  blocksize :＝ blocksize×２
 end
```

上記の手続きにおいて∧と∨はそれぞれ語のビットごとのＡＮＤ演算とＯＲ演算である．また，mask0[k]とmask1[k]は2^{k-1}個の１と2^{k-1}個の０が交互に並んだ語であり，¬mask0[k]と¬mask1[k]はそれぞれmask0[k]とmask1[k]の１と０を逆にしたものである．

source0 と source1, mask0 と mask1, shift0 と shift1, destination0 と destination1 が個々の回転等の算法を特徴づける．表**1** の各算法に対応する source0 と source1, mask0 と mask1, shift0 と shift1, destination0 と destination1の値を表**3**に示す（恒等変換Ｉは何もしないに等しく，原マトリクスと上下に対称なマトリクスを導くＶは上記の算法より高速な算法があるが，一応，ここではＩとＶも上記の算法の枠の中で論ずることにする）．

source0 と source1, mask0 と mask1, shift0 と shift1, destination0 と destination1は配列であり，第１要素，第２要素，…，第m要素は皆違うものであるが，表１の算法それぞれの中では同様の値をとるので表３には第ｋ要素だけを代表として示してある（例えば，source0[k]の値が表**3**で2^{k-1}ならば，source0[1]＝2^0，source0[2]＝2^1，…，source0[m]＝2^{m-1}と解釈する）．

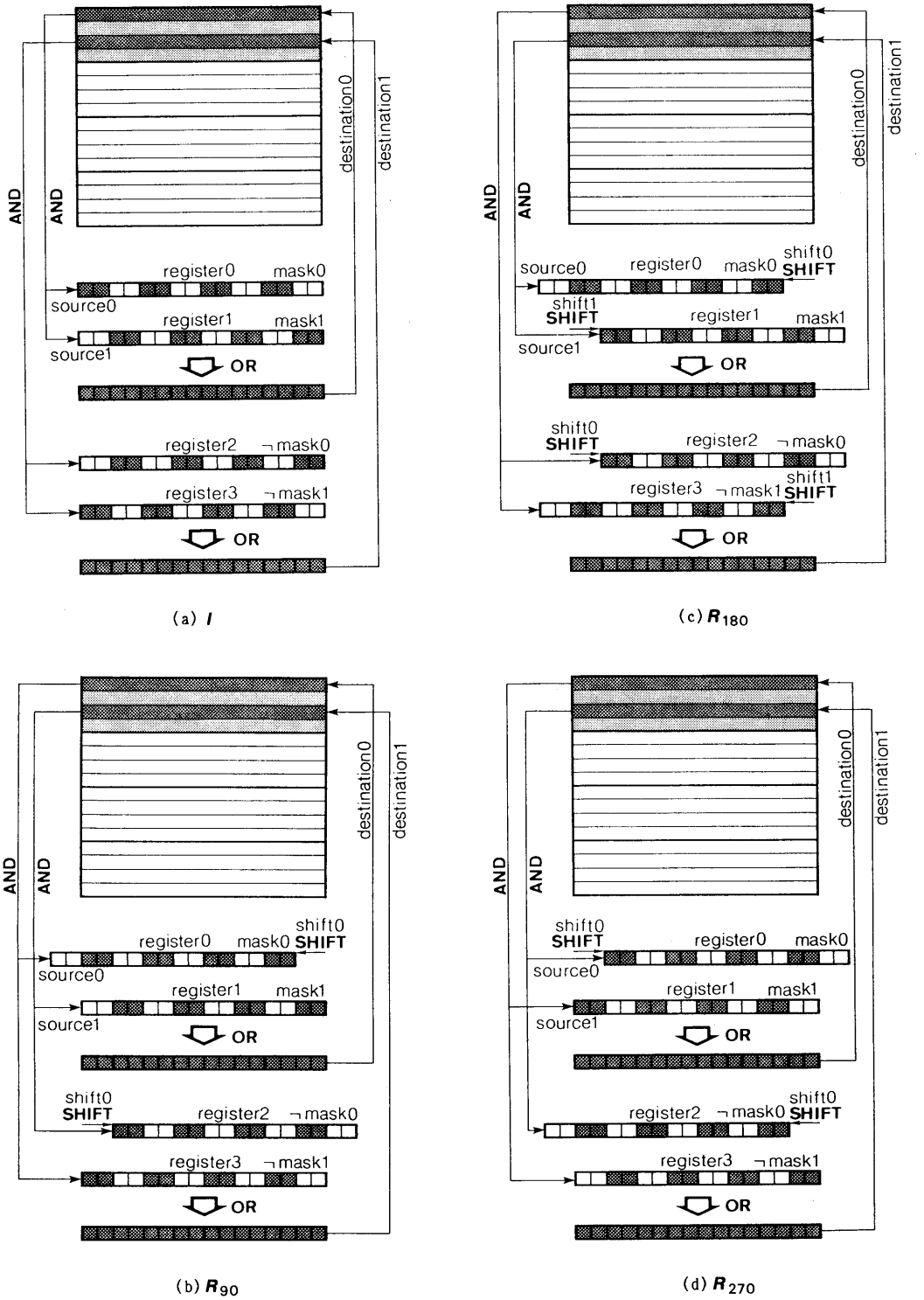
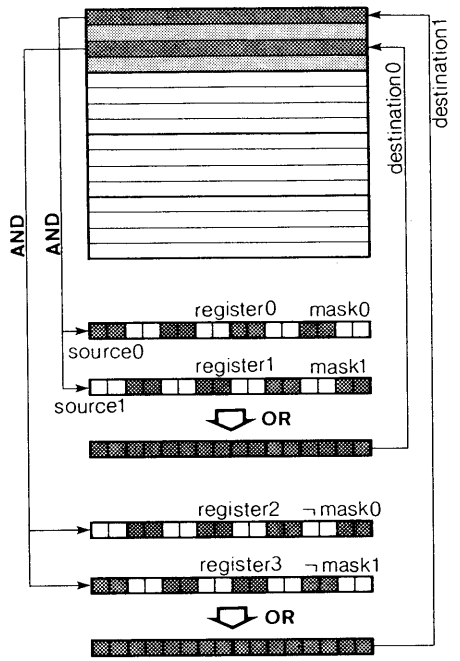
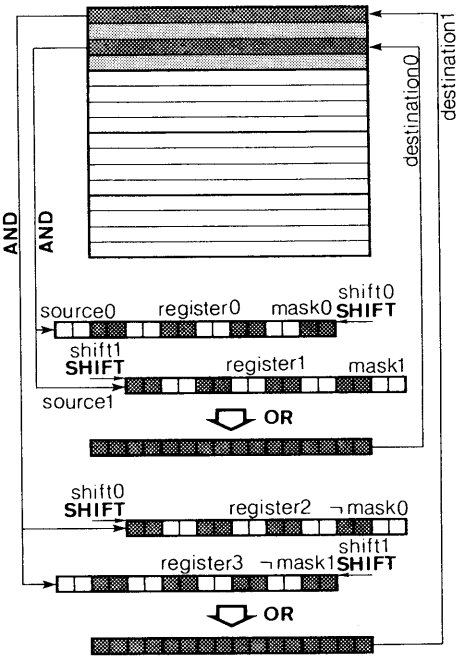


図2　各算法における部分回転の方法

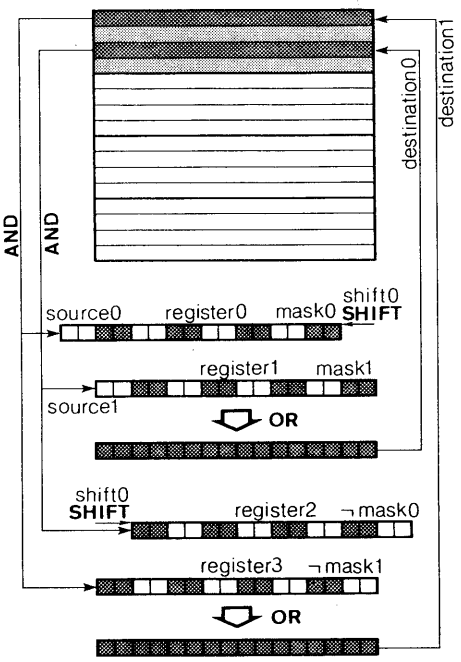
Fig. 2　Subrotation in algorithms



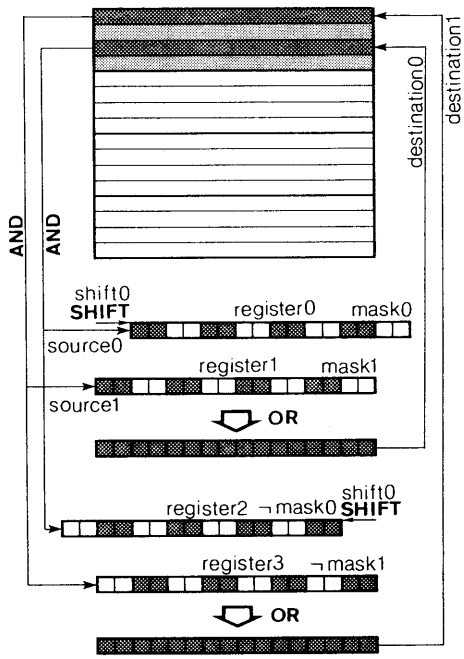
(e) ***V***



(g) ***H***



(f) ***G***



(h) ***D***

図２　各算法における部分回転の方法（続き）

Fig. 2　Subrotation in algorithms (continued)

表3　回転等の各算法の特徴付け

I	R_{90}	R_{180}	R_{270}
0/0	0/2^{K-1}	2^{K-1}/2^{K-1}	2^{K-1}/0
type1/type0	type0/type0	type0/type1	type1/type1
0/0	2^{K-1}/0	2^{K-1}/2^{K-1}	0/2^{K-1}
0/2^{K-1}	0/2^{K-1}	0/2^{K-1}	0/2^{K-1}
V	**G**	**H**	**D**
0/0	0/2^{K-1}	2^{K-1}/2^{K-1}	2^{K-1}/0
type1/type0	type0/type0	type0/type1	type1/type1
0/0	2^{K-1}/0	2^{K-1}/2^{K-1}	0/2^{K-1}
2^{K-1}/0	2^{K-1}/0	2^{K-1}/0	2^{K-1}/0

各算法について，上から順に
source0[K]/source1[K]
mask0[K]/mask1[K]
shift0[K]/shift1[K]
destination0[K]/destination1[K]
を示す．ただし，

type0: $\underbrace{0\cdots0}_{K-1}\underbrace{1\cdots1}_{K-1}\cdots\cdots\underbrace{0\cdots0}_{K-1}\underbrace{1\cdots1}_{K-1}$

type1: $\underbrace{1\cdots1}_{K-1}\underbrace{0\cdots0}_{K-1}\cdots\cdots\underbrace{1\cdots1}_{K-1}\underbrace{0\cdots0}_{K-1}$

表4　回転等の各算法の別の特徴付け

I'	R_{90}'	R_{180}'	R_{270}'
2^{K-1}/2^{K-1}	0/2^{K-1}	0/0	2^{K-1}/0
type1/type0	type1/type1	type0/type1	type0/type0
0/0	0/2^{K-1}	2^{K-1}/2^{K-1}	2^{K-1}/0
2^{K-1}/0	2^{K-1}/0	2^{K-1}/0	2^{K-1}/0
V'	**G'**	**H'**	**D'**
2^{K-1}/2^{K-1}	0/2^{K-1}	0/0	2^{K-1}/0
type1/type0	type1/type1	type0/type1	type0/type0
0/0	0/2^{K-1}	2^{K-1}/2^{K-1}	2^{K-1}/0
0/2^{K-1}	0/2^{K-1}	0/2^{K-1}	0/2^{K-1}

上記の算法で for u:=… の部分は原マトリクスにおける各$2^k\times2^k$のブロックに対する回転等（これを“部分回転等”と呼ぶ）に相当する．

図2に各回転等における部分回転の実行方法を示す．for k:=… のループがkのある値について終了すると，与えられたn×nマトリクスは各$2^k\times2^k$部分マトリクスの内部で９０°，１８０°回転するから，k＝mまで終了すると，与えられたn×nマトリクスは全体として９０°，１８０°回転する．

回転等の算法の記述は一意的とは限らない．表１の算法は表４のように記述することも可能である．表１，４の対応する算法同士はドットマトリクスに対して同じ効果をもたらすが，記述は（字面の上では）異なるものである．以下では，表１，４の計１６の算法を別のものとして扱う．

3．回転等の算法の相互変換

表１，４から観察されるように，表１，４の回転等の算法の違いは source0, source1, mask0, mask1, shift0, shift1, destination0 と destination1の違いだけであるので，簡単な規則により相互に変換することができる．次の変換規則を考える（記号′がついているのは変換後の値という意味である）．

v変換：　destination0′[k]
　　＝ 2^k－destination0[k]，
　　destination1′[k]
　　＝ 2^k－destination1[k]．

v′変換：source0′[k]＝ 2^k－source1[k]，
　　source1′[k]＝ 2^k－source0[k]，
　　mask0′[k]＝¬mask1[k]，
　　mask1′[k]＝¬mask0[k]，
　　shift0′[k]＝shift1[k]，
　　shift1′[k]＝shift0[k]，

h変換：　source0′[k]＝ 2^k－source1[k]，
　　source1′[k]＝ 2^k－source0[k]，
　　mask0′[k]＝¬mask0[k]，
　　mask1′[k]＝¬mask1[k]，
　　shift0′[k]＝2^{k-1}－shift0[k]，

shift1′ [k]＝2^{k-1}－shift1[k]，

destination0′ [k]

＝ 2^{k}－destination0[k]，

destination1′ [k]

＝ 2^{k}－destination1[k]．

h′ 変換：source0′ [k]＝ source1[k]，

source1′ [k]＝ source0[k]，

mask0′ [k]＝mask1[k]，

mask1′ [k]＝mask0[k]，

shift0′ [k]＝2^{k-1}－shift1[k]，

shift1′ [k]＝2^{k-1}－shift0[k]，

r 変換： source0′ [k] ＝ source1[k]，

source1′ [k] ＝ 2^{k}－source0[k]，

mask0′ [k] ＝ mask1[k]，

mask1′ [k] ＝ ¬mask0[k]，

shift0′ [k] ＝ 2^{k-1}－shift1[k]，

shift1′ [k] ＝ shift0[k]．

これらの変換の二つｘとｙをこの順に行なう合成変換をｙｘと記す．この記法によるｒｖをｇ変換，ｒｖ′をｇ′変換，ｒｈをｄ変換，ｒｈ′をｄ′変換と呼び，何もしない変換を１と記す．

4．変換群の構造

容易に確かめられるように，回転等の算法に対する変換は合成に関して結合律をみたし，半群をなす．この半群は複雑であるが，ｖ，ｈ，ｇ，ｄはＩ，R_{90}，R_{180}，R_{270}とＶ，Ｇ，Ｈ，Ｄとの間の（またはＩ′，R_{90}'，R_{180}'，R_{270}'とＶ′，Ｇ′，Ｈ′，Ｄ′との間の）変換であり，ｖ′，ｈ′，ｇ′，ｄ′はＩ，R_{90}，R_{180}，R_{270}とＶ′，Ｇ′，Ｈ′，Ｄ′との間の（またはＩ′，R_{90}'，R_{180}'，R_{270}'とＶ，Ｇ，Ｈ，Ｄとの間の）変換であるので，部分半群

Ｔ＝｛１，ｒ，r^2，r^3，ｖ，ｇ，ｈ，ｄ｝

と

Ｔ′＝｛１，ｒ，r^2，r^3，

ｖ′，ｇ′，ｈ′，ｄ′｝

の構造を調べることにする．合成された変換は無数にあるが，各部分半群においては実質的に相異なるものはそれぞれ上記の８つであり，これらを演算子と呼ぶことにする．

各部分半群Ｔ，Ｔ′における演算子間の合成は表**5**，**6**の通りで，逆元が存在するから，それぞれ群となる．群Ｔは表**5**の通り，可換群であり，自乗すると単位

表５　回転等の算法に対する演算子の群表

y＼x	1	r	r^2	r^3	v	g	h	d
1	1	r	r^2	r^3	v	g	h	d
r	r	r^2	r^3	1	g	h	d	v
r^2	r^2	r^3	1	r	h	d	v	g
r^3	r^3	1	r	r^2	d	v	g	h
v	v	g	h	d	1	r	r^2	r^3
g	g	h	d	v	r	r^2	r^3	1
h	h	d	v	g	r^2	r^3	1	r
d	d	v	g	h	r^3	1	r	r^2

演算子**x**に引き続いて演算子**y**を実行したときの結果

表６　回転等の算法に対する演算子の群表

y＼x	1	r	r^2	r^3	v′	g′	h′	d′
1	1	r	r^2	r^3	v′	g′	h′	d′
r	r	r^2	r^3	1	d′	v′	g′	h′
r^2	r^2	r^3	1	r	h′	d′	v′	g′
r^3	r^3	1	r	r^2	g′	h′	d′	v′
v′	v′	g′	h′	d′	1	r	r^2	r^3
g′	g′	h′	d′	v′	r^3	1	r	r^2
h′	h′	d′	v′	g′	r^2	r^3	1	r
d′	d′	v′	g′	h′	r	r^2	r^3	1

元Ｉになる元がＩ以外に３つある．群Ｔ′は表６の通り，非可換であり，自乗すると単位元１になる元が１以外に５つある．表１の回転等の各算法に上記の演算子を施した結果を図３，４に示す．回転等の諸算法のなす群は群Ｔ′と同型であるが，群Ｔとは同型でない．

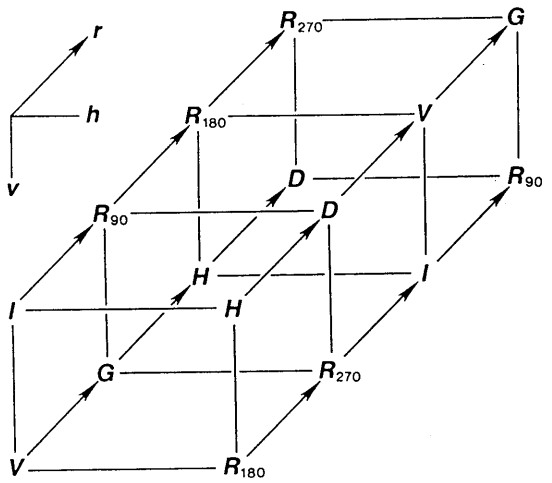


図３　回転等の算法に演算子を作用させた結果(1)

Fig.3　Result of operators on rotation algorithms (1)

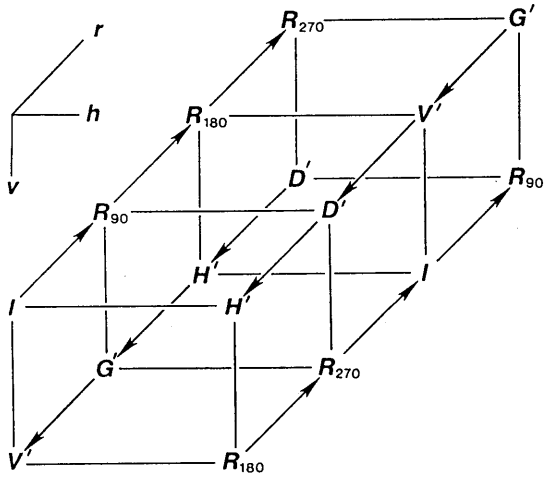


図４　回転等の算法に演算子を作用させた結果(2)

Fig.4　Result of operators on rotation algorithms (2)

６．まとめ

ドットマトリクスの回転等の算法に対する書換え演算子のなす群の構造を論じ，回転等のなす群との違いを示した．

謝辞

本研究のきっかけを与えて下さった東京農工大学工学部数理情報工学科中川正樹助教授，同大学院生関口治氏（現㈱日立製作所），有益なヒントを賜った日本電装㈱赤堀一郎氏に感謝申し上げます．本研究の一部は(財)大川情報通信基金の昭和６３年度助成を受けた．

参考文献

(1) H. Samet: “The quadtree and related hierarchical data structure”, ACM Computing Surveys **16**, 187-260 (1984).

(2) 中森眞理雄：“ドットパターンを回転するアルゴリズム”，信学論(A)，J72-A，2，444-446 (1989).

(3) A.Goldberg and D.Robson, Smalltalk-80: The Language and Its Implementation, Addison-Wesley, 1983

(4) 里山元章，中川正樹，高橋延匡：“日本語文書出力システム「浄書」の基本設計と開発システムの実現”，情報処理学会「ヒューマンフレンドリなシステム」シンポジウム報告集，pp.181-193(1986-07).

(5) 池田裕治，伊藤幸雄，真鍋俊彦，阿刀田央一，高橋延匡：“レーザビームプリンタのインテリジェントインターフェース／コントロールの試作”，情報処理学会第24回全国大会講演論文集，pp.1045-1046 (1982).

(6) 関口治，中川正樹，高橋延匡：“インテリジェント機器のシステム開発に関する一考察：レーザビーム・プリンタのインテリジェント化を例にとって”，情報処理学会第30回全国大会講演論文集，pp.671-672(1985).