覆面算を解析するためのオートマトン理論的アプローチ

遠藤　洋[†]　成澤和志[†]　篠原　歩[†]

覆面算とは，本来数字である部分が記号として与えられた等式から，その数字を探し当てる数学パズルの一種である．本稿では，覆面算全体を網羅的に考察するために，正しい覆面算を受理するオートマトンの構造について考察する．また，2 進数，3 進数，4 進数の覆面算に対して，実際に生成した決定性オートマトンの状態数について報告する．

An Automaton Theory Approach for Analyzing Alphametic

HIROSHI ENDOH ,[†] KAZUYUKI NARISAWA [†] and AYUMI SHINOHARA [†]

Alphametic is a puzzle which is included in the mathematical puzzle. Players are given a mathematical formula which replaced number by symbol. In this paper, we consider about automaton which accept correct alphametic to consider about hole of t alphametic cyclopaedically. We also report the number of state of deterministic automaton which is generated actually in case of alphametics for binary number, 3-adic number and 4-adic number.

1. はじめに

覆面算 (alphametics または cryptarithm) とは，計算式の 0〜9 の各数字が，それぞれ別の記号に置き換えられた形で与えられたとき，どの記号がどの数字に対応しているかを推測し，元の完全な計算式を導き出すパズルである．図 1 は，H.E.Dudeney[1),2)] による有名な覆面算の例であり，図 2 はその解である．

	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

図 1　覆面算の例

	9	5	6	7
+	1	0	8	5
1	0	6	5	2

図 2　例の解

覆面算が与えられたとき，ある記号から数字への置き換え規則が存在して，置換後の計算式が成り立っているとき，与えられた覆面算を**正しい覆面算**と呼ぶ．覆面算は，数字に対応する記号を変えることにより，1 つの計算式から多くの解を導くことができる．そこで，本質的に同じ覆面算（1 つの計算式から導かれた解）は 1 つとしたとき，任意の大きさに対する正しい覆面算を数え上げ，数独プロジェクト[3)] のように各覆面算に対して番号付を行うことが，本研究の最終目標である．

計算式が k 進数となる長さ n の覆面算が与えられたとき，正しいかどうかの判定は，すべての数字の順列を列挙して与えられたパターンに対応付け，その順列が正しい計算かどうか確かめることで $O(k!n)$ で行うことができる[4)]．一方，k も入力として与えられる変数と見なしたとき，答えを見つけ出す問題は NP 困難になることがわかっている[5)]．

ここで，覆面算の解が唯一のとき，その覆面算は**純**であるという．覆面算が純であるかの判定も $O(k!n)$ で計算できる．すなわち，入力長 n に対しては線形時間で解けることになる．しかしながら，この定数 $k!$ が極めて大きいため，実用的とは言えない．そこで，本研究では k 進数-覆面算に対して有限オートマトンの理論によるアプローチを行う．決定性オートマトンを構築しておけば，覆面算は実時間で解くことが可能になるが，状態数は極めて大きいと予想される．本論文では，いくつかの k に対して実際にオートマトンを構築する．このオートマトンを解析することで，覆面算の番号付にも応用できると考えられる．

本論文では，まず覆面算を形式的に定義する．また，最も単純な覆面算の系列にある変換操作を加えることで正しい覆面算の集合が正規言語になることを示し，正しい覆面算系列のみを受理する有限決定性オートマトンを作成する．

† 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University

2. 定　　義

2.1 覆 面 算

$k \in \mathbb{N}$ に対する記号の集合を $A_k = \{a_0, a_1, \ldots, a_{k-1}\}$，数値の集合を $N_k = \{n \in \mathbb{N} \mid 0 \leq n \leq k-1\}$ とする．

定義 1 (k 進数-覆面算)**.** 系列 $w_1, w_2, \ldots, w_{n-1}, w_n \in A_k^*$ に対して，$w_1 \oplus w_2 \oplus \ldots \oplus w_{n-1} = w_n$ という形の式を**覆面算**と呼ぶ．ここで，$\oplus$ は任意の演算子である．次の条件を満たす単射 $f : A_k \to N_k$ をこの覆面算の**解**と呼ぶ．

(1) f によって定まる関数 $g : {A_k}^* \to N$ があったとき，覆面算の各文字を数字に置き換えて得られる式 $g(w_1) \oplus g(w_2) \oplus \ldots \oplus g(w_{n-1}) = g(w_n)$ が成り立つ．ここで，関数 g は入力系列の各文字を関数 f によって置き換え，k 進数の数を出力する．

(2) 各 w_i の先頭文字 a に対し，$f(a) \neq 0$.

解となる単射 f が存在しないとき，この覆面算は**解けない**と言う．

解を持つ覆面算を**正しい覆面算**と呼ぶ．正しい k 進数-覆面算全体からなる集合を k **進数-覆面算言語**と呼び，以降 R_k と記す．

一般に各項 w_i の長さは異なるが，上位桁に記号 $\$ \notin \Sigma$ を必要なだけ補って項長を揃え，さらにもう一つずつ記号 $ を区切り記号として追加した式を考える．この覆面算を**等項長覆面算**と言う．正しい k 進数-等項長覆面算の集合を k **進数-等項長覆面算言語**と言い，R_k^{term} と記す．等項長覆面算を筆算の形にし最下位の桁から縦に読みながら連結してできる系列を**正規覆面算**と言う．解をもつ k 進数-正規覆面算の集合を k **進数-正規覆面算言語**と言い，$R_k^{regular}$ と記す．

例えば，図 1 の覆面算に対する等項長覆面算は $$SEND+$$MORE=$MONEY であり，また，正規覆面算は DEYNREEONSMO$$M$$$ である．

k 進数-覆面算言語 R_k，k 進数-等項長覆面算言語 R_k^{term}，k 進数-正規覆面算言語に対し，項数が m である部分言語をそれぞれ (k,m)-覆面算言語 $R_{k,m}$，(k,m)-等項長覆面算言語 $R_{k,m}^{term}$，(k,m)-正規覆面算言語 $R_{k,m}^{regular}$ と呼ぶ．

解となる単射が複数ある場合，この覆面算は**複数解をもつ**と言う．また，同一言語上の覆面算系列において，異なる系列 w と w' に対して，ある単射 $h : A_k \to A_k$ が存在して $w' = h(w)$ となるとき，系列 w と w' は**等価**であると言う．

定義 2. 通常の等号記号，連結記号と区別するために，系列で使用する等号記号を $=_s$，加算記号を $+_s$ と表す．また，k 進数-覆面算系列 u が与えれたとき，$u = x =_s y$ と書ける．この時，x を系列 u における左辺，y を右辺と呼ぶ．

以下，演算子 $\oplus$ を加算 $+$ に限定し，項数 m は定数とする．

以降，定数 k は任意の進数を表すものとする．また，オートマトン M を $M = (Q, \Sigma, \delta, q, F)$ と表現したとき，Q は状態集合，Σ はアルファベット，δ は遷移関数，$q \in Q$ は初期状態，$F \subseteq Q$ は受理状態集合を表す．

2.2 オートマトンの合成

定義 3. 2 つのオートマトン，$M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$，$M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ が与えられたとき，合成関数 $\mathrm{union}(M_1, M_2)$ を次のように定義する．

$$\mathrm{union}(M_1, M_2) = (Q_1 \times Q_2, \Sigma, \delta_\cap, (q_0^1, q_0^2), F_\cap)$$

ここで，状態遷移関数 $\delta_\cap$ は

$$\delta_\cap((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

であり，受理状態 $F_\cap$ は

$$F_\cap = \{(f_1, f_2) \mid f_1 \in F_1 \text{かつ } f_2 \in F_2\}$$

である．これは，オートマトン M が受理する言語を $L(M)$ と表せば，

$$L(\mathrm{union}(M_1, M_2)) = L(M_1) \cup L(M_2)$$

を意味している．以降，$\mathrm{union}(M_1, M_2)$ を記号 $\times_\cup$ を用いて $M_1 \times_\cup M_2$ と表す．

定義 4. 2 つのオートマトン，$M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$，$M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ が与えられたとき，合成関数 $\mathrm{intersect}(M_1, M_2)$ を次のように定義する．

$$\mathrm{intersect}(M_1, M_2) = (Q_1 \times Q_2, \Sigma, \delta_\cup, (q_0^1, q_0^2), F_\cup)$$

ここで，状態遷移関数 $\delta_\cup$ は

$$\delta_\cup((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

であり，受理状態 $F_\cup$ は

$$F_\cup = \{(f_1, f_2) \mid f_1 \in F_1 \text{または } f_2 \in F_2\}$$

である．これは，オートマトン M が受理する言語を $L(M)$ と表せば，

$$L(\mathrm{intersect}(M_1, M_2)) = L(M_1) \cap L(M_2)$$

を意味している．以降，$\mathrm{intersect}(M_1, M_2)$ を記号 $\times_\cap$ を用いて $M_1 \times_\cap M_2$ と表す．

正規言語のクラスは和集合，共通集合の元で閉じているため，得られる結果は演算の順番に依存しない．そこで，複数のオートマトン $M_1, M_2, \cdots, M_n$ に対して，union による直積 $M_1 \times_\cap M_2 \times_\cap \cdots \times_\cap M_n$ を $\bigcap_{1 \leq i \leq n} M_i$ と表し，intersect による直積 $M_1 \times_\cup M_2 \times_\cup \cdots \times_\cup M_n$ を $\bigcup_{1 \leq i \leq n} M_i$ と表す．

ここで，新たに二つのオートマトン合成の演算を定義する．

定義 5 (条件付き合成)**.** 複数のオートマトン $M_1, M_2, \cdots, M_n$ $(M_i = (Q_i, \Sigma, \delta_i, q_{s_i}, F_i))$ と条件となるオートマトン

$M_d=(Q_d,\Sigma,\delta_d,q_{s_d},F_d)$ の条件付合成オートマトン M_p を

$$M_p=(Q_p,\Sigma,\delta_p,((q_{s_1},\ldots,q_{s_n}),q_{s_d}),F_p)$$

とする．ここで，

$$Q_p=(Q_1\times\cdots\times Q_n)\times Q_d$$

であり，状態遷移関数 δ_p は

$$\begin{aligned}&\delta_p(((q_1,\cdots,q_n),q_d),a)\\&=((\delta_1(q_1,a),\cdots,\delta_n(q_n,a)),\delta_d(q_d,a))\end{aligned}$$

である．また，$(q_1,q_2,\cdots,q_n)\in Q_1\times Q_2\times\ldots\times Q_n$ に対して，それぞれの状態が元のオートマトンで受理状態である状態数を返す関数を $P:Q_1\times Q_2\times\ldots\times Q_n\to N$，$h_d:Q_d\to N$ を M_d の状態に応じて決まった数を出力する関数とするとき，M_p の受理状態集合 F_p は，

$$F_p=\left\{(q_1,\cdots,q_n,q_d)\;\middle|\;\begin{array}{c}P((q_1,\cdots,q_n))\le h_d(q_d)\\ \text{かつ } q_d\in F_d\end{array}\right\}$$

である．この合成によって得られるオートマトン M_p を $\biguplus_{1\le i\le n}(M_i,M_d)\mid P,h_d$ と表す．

この合成では，M_d の各状態に割り当てられた数に対して，合成後の状態を受理状態に含めるか判定している．

具体的な例を図 5 に示す．複数のオートマトンを M_1,M_2 とし，条件となるオートマトンを M_d とする．合成したオートマトンは状態数が多くなるため，ここでは省略する．ここで，関数 h_d を次のように定める．

$$h_d(q)\begin{cases}i & (q=q_i\text{のとき})\\ 0 & (q=q_{trash}\text{のとき})\end{cases}$$

このように h_d を定めると，次のように合成後の状態 q' が受理状態 F_p に含まれるかが決まる．

(1) $q'=((q_0,q_0),q_0)$ について
$q_0\notin F_d$ であるから，$q'\notin F_p$.

(2) $((q_2,q_1),q_2)$ について
$q_2\in F_1$，$q_1\notin F_2$ であるから，$P(((q_2,q_1),q_2))=1$．また，$h_d(q_2)=2$ より，$P(((q_2,q_1),q_2))\le h_d(q_2)$．ここで，$q_2\in F_d$ でもあるから，$q'\in F_p$.

(3) $((q_2,q_2),q_1)$ について
$q_2\in F_1$，$q_1\notin F_2$ であるから，$P(((q_2,q_1),q_2))=2$．また，$h_d(q_1)=1$ より，$P(((q_2,q_1),q_2))\ge h_d(q_2)$．したがって，$q'\notin F_p$.

定義 6 (*Zip* 関数)**.** オートマトン $M=(Q,\Sigma,\delta,q_0,F)$ と正整数 m が与えられたとき，関数 $Zip(M,m)$ はオートマトン $M_{zip}=(Q_{zip},\Sigma,\delta_{zip},q'_0,F_{zip})$ を返す．ここで，

$$Q_{zip}=\prod_m(Q\cup P)$$

であり，P は状態集合 $Q=\{q_1,q_2,\cdots,q_n\}$ と同じ大きさの状態集合 $P=\{p_1,p_2,\cdots,p_n\}$ である．また，

$$\begin{aligned}q'_0&=(q_0,q_0,\cdots,q_0),\\F_{zip}&=(q^0,q^1,\cdots,q^{m-1})\ (\forall i,q^i\in F),\\&\delta_{zip}((q^0,q^1,\cdots,q^{m-1}),a)\\&=\left\{\begin{array}{l}(q^0,\cdots,q^i,\phi(\delta(q^{i+1},a)),q^{i+2},\cdots,q^{m-1})\\ \qquad\left(\begin{array}{l}1\le i\le m-2\text{ に対して}\\ q^0,q^1,\cdots,q^i\in P\text{ かつ}\\ q^{i+1},q^{i+2},\cdots,q^{m-1}\in Q\text{ のとき}\end{array}\right)\\ (\phi(q^1),\phi(q^2),\cdots,\phi(q^{m-2}),\delta(q^{m-1},a))\\ \qquad\left(\begin{array}{l}q^0,q^1,\cdots,q^{m-2}\in P\text{ かつ}\\ q^{m-1}\in Q\text{ のとき}\end{array}\right)\end{array}\right.\end{aligned}$$

である．ただし，

$$\phi(q)=\begin{cases}p_i & (q=q_i\text{ のとき})\\ q_i & (q=p_i\text{ のとき})\end{cases}$$

である．

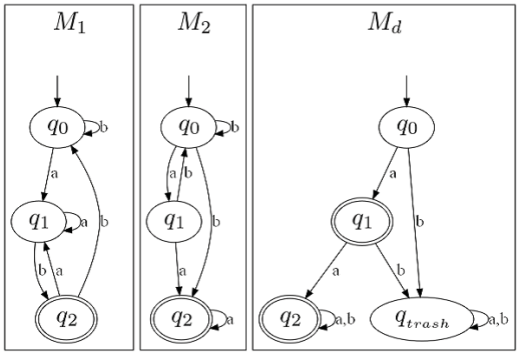

図 3 条件付き合成直積の例．M_1,M_2 が合成されるオートマトンであり，M_d が条件となるオートマトンである．

3. 定　　理

まず証明に必要な，基本的な補題を挙げる．

補題 1 (反復補題)**.** Σ 上の任意の正規言語 L に対して，次の条件を満たす正整数 l が存在する．L に属する長さが l 以上の任意の系列 w は，適当な $x,y,z\in\Sigma^*$ が存在して，$w=xyz$ を用いて，次の 3 つの条件を満たす．

(1) 任意の $i\ge 0$ に対して，$xy^iz\in L$,

(2) $|y|\ge 1$,

(3) $|xy|\le l$.

また，すべての正しい覆面算を受理するオートマトンを生成することはできないが，範囲を限定し，入力の与え方を工夫することによって生成可能になることを表す一連の定理を以下に示す．

定理 1. k 進数-覆面算言語 R_k は正規ではない．

証明. k 進数-覆面算言語 R_k が正規言語だと仮定し，背理法によって矛盾を示す．入力系列 w が R_k に含まれるとすると，$w = v_1 +_s v_2 +_s \cdots +_s v_m =_s u$ と表すことができる．また，R_k は正規言語であるため，補題 1 より系列 w は適当な $x,y,z \in \Sigma^*$ によって，$w = xyz$ と書き表すことができる．ここで，w は必ず記号 $=_s$ を含むので，x,y,z のいずれかに $=_s$ が含まれる．

(1) $=_s$ が x に含まれる場合
x は左辺と $=_s$，右辺の一部を含んでいるため，$x = u =_s v$ と表すことができる．ここで，補題 1 より，任意の $i \geq 2$ に対して，$w' = xy^i z = u =_s vy^i z \in R_k$ である．しかし，それぞれの w と w' の右辺は $vyz \neq vy^i z$ であるため，矛盾．

(2) $=_s$ が y に含まれる場合
このとき，どのような $i \geq 0$ に対しても $xy^i z \in R_k$ であるため，$w' = xz \in R_k$ である．しかし，系列 w' は記号 $=_s$ を含まないため，覆面算系列として成り立たず，矛盾．

(3) $=_s$ が z に含まれる場合
$(=_s \in x)$ の場合と同様の議論により，矛盾することが示せる．

以上のことから，k 進数-覆面算 R_k は正規言語ではない． □

定理 2. k 進数-等項長覆面算言語 R_k^{term} は正規ではない．

証明. R_k^{term} を正規言語とすると，補題 1 によって系列 $w \in R_k^{term}$ は $w = xyz$ と書き表せる．定理 1 と同様の議論により，R_k^{term} は正規言語ではない． □

定理 3. k 進数-正規覆面算言語 $R_k^{regular}$ は正規でない．

証明. $R_k^{regular}$ が正規言語であるとする．このとき $R_k^{regular}$ に対して，補題 1 の条件 3 を満たす正整数 l が存在する．ここで，任意の項数 $m(> l)$ をもつ系列 $w \in R_k^{regular}$ を考える．すると，補題 1 より適当な $x,y,z \in \Sigma^*$ があって $w = xyz$ と表すことができ，$|xy| \leq l(< m)$ が成り立つ．さらに，任意の $i \geq 0$ に対して $xy^i z \in R_k^{regular}$ であるが，i を変動させた場合，覆面算系列として成り立たたないため，矛盾．したがって，$R_k^{regular}$ は正規言語ではない． □

定理 4. (k,m)-覆面算言語 $R_{k,m}$ は正規ではない．

証明. 定理 1 と同様の議論により，証明できる． □

定理 5. (k,m)-等項長覆面算言語 $R_{k,m}^{term}$ は正規ではない．

証明. 定理 2 と同様の議論により，証明できる． □

ここで，以下 m を固定する．

定理 6. (k,m)-正規覆面算言語 $R_{k,m}^{regular}$ は正規言語である．ただし，$k > 1$ とする．

根拠となるオートマトンの構築方法を次章に述べる．

4. $R_{k,m}^{regular}$ を受理するオートマトン

(k,m)-正規覆面算言語 $R_{k,m}^{regular}$ を受理する有限オートマトンは，入力として与えられた覆面算が以下の条件を満たしているかを判定すればよい．ただし，$k = 1$ の場合は必ず各項の左端が 0 になってしまうため，成り立たない．

(1) ある写像 f にて加算が成り立っている．
(2) 各項の左端の記号 a に対して，$f(a) \neq 0$.
(3) 等価な R^{term} に変換した際，各項の中に記号 \$ が出現しない．

これらの条件を満たすオートマトンを生成できればよい．ただし，条件 (3) は通常考える必要はないが，$R^{regular}$ のみを考える場合考慮しなければいけない制約である．

まず，条件 (1)，(2)，(3) をそれぞれ満たす有限決定性オートマトン，$M_{k,m}^{(1)}$，$M_{k,m}^{(2)}$，$M_{k,m}^{(3)}$ を生成する．これらのオートマトンを合成することにより，3 つの条件を満たすオートマトン $M_{k,m}^{regular}$ が得られるが，これは等価な覆面算系列を重複して受理してしまう．そこで，等価な覆面算系列のうち 1 つだけを受理するためのオートマトン $M_{k,m}^{normal}$ を合成することにより，正しい覆面算 $R_{k,m}^{regular}$ を受理し，かつ等価な覆面算を複数受理しないオートマトンを得ることができる．

4.1 加算が正しいか判定するオートマトン

解になり得る写像 $f : \Sigma \to N_k$ が与えられているとする．このとき，写像 f を解として正しい加算についてのみ必ず成り立つ系列を受理するオートマトン $M_{k,m}^{(1)}(f)$ を示す．その前準備として，l 桁目の加算が正しいか判定するオートマトン $M_{k,m}^{add}(f,l)$ を部分的に作る．$R_{k,m}^{regular}$ の系列は一桁目から順に見ているため，このオートマトンは以下のようになる．

$$M_{k,m}^{add}(f,l) = (Q_l, \Sigma, \delta_l, q_0^0, F_l)$$

ここで，

$$Q_l = \bigcup_{0 \leq i \leq m-1} Q^i \cup Q^{ans} \cup \{q^{trash}\}$$

である．ただし，

$$Q_l^i = \bigcup_{0 \leq n \leq g(l)} \{q_n^i\},$$

$$Q^{ans} = \bigcup_{0 \leq n \leq g(l+1)} \{q_n^{ans}\}$$

であり，$g(l)$ は l 桁目における繰り上がりの最大数である．また，

$$F_l = Q^{ans},$$

$$\delta_l(q,a) = \begin{cases} a_{i+f(a)}^{n+1} & \left(\begin{array}{l} q = q_i^n \text{ かつ} \\ n \neq m-1 \text{ のとき} \end{array}\right) \\ q_{\lfloor \frac{i}{k} \rfloor}^{ans} & \left(\begin{array}{l} q = q_i^2 \text{ かつ} \\ i = f(a) \bmod k \text{ かつ} \\ a \neq \$ \text{ のとき} \end{array}\right) \\ q^{trash} & otherwise \end{cases}$$

である．

これはどの桁においても $g(l)$ が一定であれば同一の形をしている．そこで，最大の桁上げ数 $\max_{l \geq 0} g(l)$ を求める．これが存在するためには，十分大きな l が存在して，

$$g(l+1) \cdots k \leq g(l) + (m-1)(k-1)$$

が成り立てばよい．ここで，

$$g(l) < \sum_{i=1}^{l} \frac{(k-1)(m-1)}{k^i}$$

であり，$k \neq 1$ とすると

$$\lim_{l \to \infty} \sum_{i=1}^{l} \frac{(k-1)(m-1)}{k^i} = k(m-1)$$

であるため，すべての l に対して $g(l) = k(m-1)$ と定めれば，$k=1$ を除き，各系列の各桁に対して同一のオートマトンによって判定できる．この結果を用いると以下のように $M_{k,m}^{(1)}(f)$ を定義できる．

$$M_{k,m}^{(1)}(f) = (Q^{(1)}, \Sigma, \delta_f^{(1)}, q_0^{(1)}, F^{(1)})$$

ここで，

$$Q^{(1)} = \bigcup_{0 \leq i \leq m-1} Q^i \cup \{q^{trash}, q^{accept}\} \tag{1}$$

である．ただし，

$$Q^i = \bigcup_{0 \leq n \leq k(m-1)} \{q_n^i\}$$

である．また，

$$F^{(1)} = \{q^{accept}\},$$

$$\delta_f^{(1)}(q,a) = \begin{cases} a_{i+f(a)}^{n+1} & \left(\begin{array}{l} q = q_i^n \text{ かつ} \\ n \neq (m-1) \text{ のとき} \end{array}\right) \\ q_{\lfloor \frac{i}{k} \rfloor}^{0} & \left(\begin{array}{l} (q = q_i^{k-1} \text{ かつ} \\ i = f(a) \bmod k \text{ かつ} \\ a \neq \$ \text{ のとき} \end{array}\right) \\ q^{accept} & \left(\begin{array}{l} q = q_0^{k-1} \text{ かつ} \\ a = \$ \text{ のとき} \end{array}\right) \\ q^{trash} & otherwise \end{cases}$$

である．正規覆面算系列の定義より，受理状態の直前に読み込む記号は必ず記号 \$ であるため，\$ を読み込んだときのみ受理状態へ遷移する．具体的な例として，$k=3, m=3$ の場合に生成されるオートマトン $M_{3,3}^{(1)}$ を図 4 に示す．

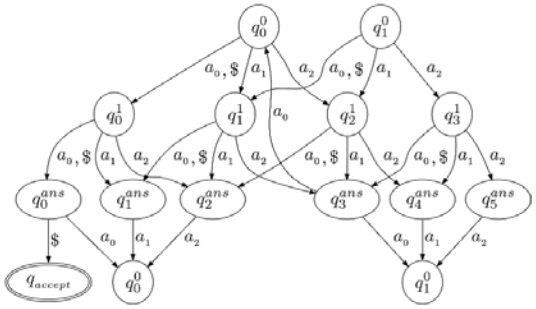

図 4 $M_{3,3}^{(1)}$ の例．ただし，図が煩雑になるのを防ぐため，状態 q_{trash} とそれへの遷移は全て省略した．また，図内上にある状態 q_0^0, q_1^0 と，図内下にある状態 q_0^0, q_1^0 は同じ状態を表している．

4.2 形式的な定義の条件を満たしているか判定するオートマトン

4.1 と同様，解になり得る写像 $f : \Sigma \to N_k$ が与えられているとする．この写像 f を解として，条件 (2) を満たすオートマトン $M_{k,m}^{(2)}(f)$ と条件 (3) を満たすオートマトン $M_{k,m}^{(3)}(f)$ を生成する．

しかし，これらのオートマトンは形式的に生成することが難しい．そこで，正規覆面算言語を入力とするオートマトンの代わりに，各項の逆系列を入力とする小規模なオートマトンをそれぞれ生成する．その後，これらの共通直積を取り関数 Zip を適用すれば，結果として $M_{k,m}^{(2)}(f) \times_{\cap} M_{k,m}^{(3)}(f)$ と等価なオートマトンが生成できる．

条件 (2) に対する小規模なオートマトン $M_k^{2'}$ は，

$$M_k^{2'}(f) = (Q_2, \Sigma, \delta_2, q_0, F_2)$$

である．ここで，

$$Q_2 = \{q_0, q^{accept}\}, \tag{2}$$

$$F_2 = \{q^{accept}\},$$

$$\delta_2(q,a) = \begin{cases} q_0 & \left(\begin{array}{c} q = q_0 \text{かつ } f(a) = 0 \\ \text{または} \\ q = q^{accept} \text{かつ } f(a) = 0 \\ \text{のとき} \end{array}\right) \\ q^{accept} & \left(\begin{array}{c} otherwise \end{array}\right) \end{cases}$$

である．具体的な例として，$k=3$ の場合に生成されるオートマトン $M_3^{2'}$ を図 5 に示す．

また条件 (3) に対する小規模なオートマトン $M_k^{3'}$ は，

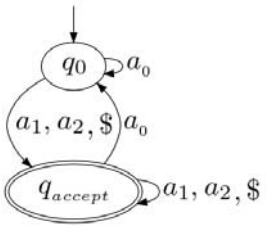

図 5 $M_3^{3'}$ の例

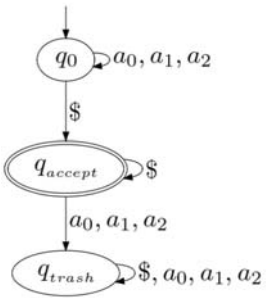

図 6 $M_3^{3'}$ の例.
ただし，$f(a_0)=0$ とした場合

$$M_k^{3'}(f)=(Q_3,\Sigma,\delta_3,q_0,F_3)$$

である．ここで，

$$Q_3=\{q_0,q^{accept},q^{trash}\}, \tag{3}$$

$$F_3=\{q^{accept}\},$$

$$\delta_3(q,a)=\begin{cases} q_0 & \left(\begin{array}{c} q=q_0\text{かつ } a\neq\$ \\ \text{のとき}\end{array}\right) \\ q^{accept} & \left(\begin{array}{c} q=q_0\text{かつ } a=\$ \\ \text{または} \\ q=q^{accept}\text{かつ } a=\$ \\ \text{のとき}\end{array}\right) \\ q^{trash} & (otherwise)\end{cases}$$

である．具体的な例として，$k=3$ の場合に生成されるオートマトン $M_3^{3'}$ を図 6 に示す．

これらを和直積によって合成したオートマトンを，項数 m によって変換する．

$$M_{k,m}^{def}(f)=Zip((M_{k,m}^{2'}(f)\times_{\cap}M_{k,m}^{3'}(f)),m) \tag{4}$$

これにより，条件 (2)，(3) を満たすオートマトンが得られた．

4.3 入力の正規化判定を行うオートマトン

以上の手順により，写像 f に対応した条件 (1)〜(3) を満たすオートマトン $M_{k,m}^{(1)}(f)\times_{\cap}M_{k,m}^{def}(f)$ が得られた．しかし，これらは等価な覆面算系列を重複して受理してしまう．例えば，$\Sigma=\{a_0,a_1,a_2,+_s,=_s\}$ として，以下の例が挙げられる．

$a_0+_sa_0=_s a_1$,
$a_0+_sa_0=_s a_2$,
$a_1+_sa_1=_s a_0$,
$a_1+_sa_1=_s a_2$,
$a_2+_sa_2=_s a_0$,
$a_2+_sa_2=_s a_1$

これらの系列は一意の変換により，次の系列に統一できる．

$a_0+_sa_0=_sa_1$

これらを 1 つとして数えるために，入力の正規化判定を行う以下のオートマトン M_k^{normal} を考える．

$$M_k^{normal}=(Q^{normal},\Sigma,\delta^{normal},q_0,Q^{normal}\setminus\{q^{trash}\})$$

ここで，

$$Q^{normal}=\{q_0,q_1,\ldots,q_k,q^{trash}\}, \tag{5}$$

$$\delta^{normal}(q_i,a_t)=\begin{cases} q_i & \left(\begin{array}{ll} t<i & \text{または} \\ a_t=\$ & \text{のとき}\end{array}\right) \\ q_{i+1} & (t=i\text{のとき}) \\ q^{trash} & \left(\begin{array}{ll} t>i & \text{または} \\ q_i=q^{trash} & \text{のとき}\end{array}\right)\end{cases}$$

である．具体的なオートマトン M_3^{normal} を図 7 に示す．

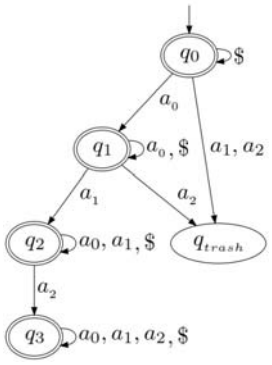

図 7 M_3^{normal} の例

M_k^{normal} を合成することにより，正しい覆面算 $R_{k,m}^{regular}$ を受理し，かつ等価な覆面算は複数受理しないオートマトン

$$M_{k,m} = M_m^{normal} \times_\cap \left(\bigcup_{f \in Func(k)} (M_{k,m}^{(1)}(f) \times_\cap M_{k,m}^{def}(f)) \right)$$

が得られる．ここで，$Func(k)$ は全ての解の集合である．

4.4 複数解を受理しないオートマトン

さて，これまで述べたオートマトンでは正規な覆面算か，正規でない覆面算かの判定しか行えず，純であるか，そうでないかの判定は行えなかった．そこで，合成の演算を変更する．

オートマトン $M_{k,m}^{normal}$ の状態 Q^{normal} に対して，関数 h_d を

$$h_d(q) = \begin{cases} 0 & (q = q_{trash} \text{のとき}) \\ 1 & (q = q_k \text{のとき}) \\ (k-i-1)! & (q = q_i \text{のとき}) \end{cases}$$

と定める．これを用いて，

$M_{k,m}^{pure} = \biguplus_{f \in Func(k)} ((M_{k,m}^{(1)}(f) \times_\cap M_{k,m}^{def}(f), M_k^{normal}) | P, h_d$

と定めると，オートマトン $M_{k,m}^{pure}$ が受理する言語は，k 進数における m 項の，純である正規な覆面算を受理する．

5. 実　　験

5.1 進数 k と項数 m によるオートマトンの状態数

合成したオートマトン $M_{k,m}$ と $M_{k,m}^{\#}$ の状態数を実際に表してみる．ただし，関数 $size(M)$ はオートマトン M の状態数を表す．

式 (1) より，

$$\begin{aligned} size(M_{k,m}^{(1)}) &= |Q^{(1)}| \\ &= |\bigcup_{0 \le i \le (m-1)} Q^i \cup \{q^{trash}, q^{accept}\}| \\ &= |\bigcup_{0 \le i \le k(m-1)} \{q_n^i\} \cup \{q^{trash}, q^{accept}\}| \\ &= (k(m-1) \times m) + 2 \end{aligned}$$

式 (2),(3),(4) より，

$$\begin{aligned} size(M_{k,m}^{def}) &= size(Zip((M^2 \times M^3), m)) \\ &= (size(M^2) \times size(M^3))^m \\ &= (3 \times 2)^m \\ &= 6^m \end{aligned}$$

式 (5) より，

$$\begin{aligned} size(M_k^{normal}) &= |Q_k^{normal}| \\ &= |\{q_0, q_1, \cdots, q_k, q^{trash}\}| \\ &= k+2 \end{aligned}$$

したがって，

$$size(M_{k,m}) = ((2 + k(m-1)m) \times 6^m)^{k!} \times (k+2)$$

表 1　項数 $m = 3$，$k = 2,3,4$ に対するオートマトンの状態数．状態数 (1) は純な覆面算系列のみ受理する場合，状態数 (2) は純でない覆面算系列も受理する場合である．

k	$M_{k,3}^{(1)}$	$M_{k,3}^{def}$	$M_{k,3}^{pure}$	$M_{k,3}$
2	10	81	120	246
3	13	81	150	1,316
4	16	81	354,565	354,565

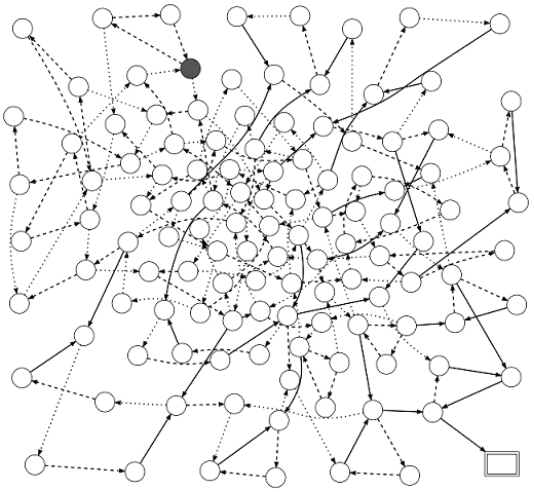

図 8　(2,3)-正規覆面算言語 $R_{2,3}^{regular}$ の純なもののみ受理するオートマトンである．黒丸が開始状態であり，四角が受理状態である．状態遷移を表す線の種類の違いは記号の種類の違いである

また，演算 $\biguplus$ でも状態数は変化しないため，

$$size(M_{k,m}^{pure}) = ((2 + k(m-1)m) \times 6^m)^{k!} \times (k+2)$$

ただし，実際には到達不可能な状態が存在したり，状態数の最小化を施すことで少なくなったりするため，この式よりも小さくなる．

5.2 k による有限オートマトンの状態数

実際に，$m = 3, k = 2,3,4$ における各オートマトンを作成し，状態数を数えた．その結果を表 1 に示す．この結果から，k によって状態数が爆発的に増加していることがわかる．また，(2,3)-正規覆面算言語の純なもののみを受理するオートマトンを図 8 に，(2,3)-正規覆面算言語を受理するオートマトンを図 9 に示す．

6. ま と め

本論文では，覆面算の問題を言語の問題として扱い，それらを受理するオートマトンとして表した．その準備として，覆面算を単純な系列として扱うと正規言語には含まれないが，進数 k と項数 m を固定し，特殊な変換操作を適用すれば正規言語に含まれることを示した．

また，進数 k と項数 m を定めた場合のオートマト

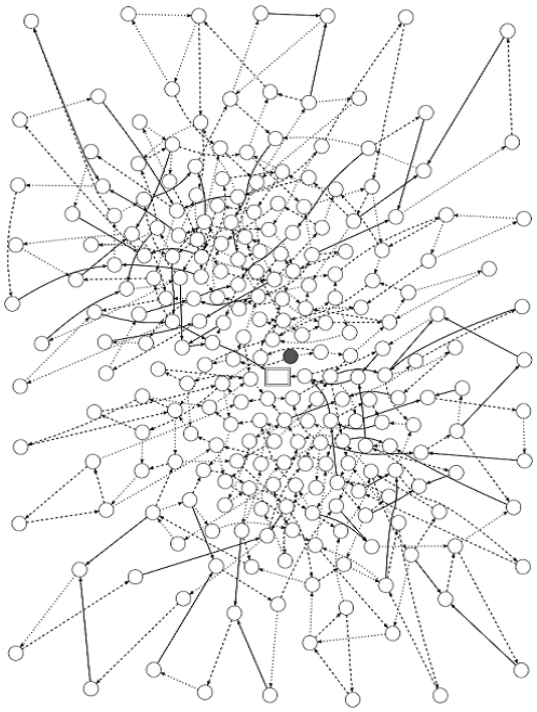

図 9 (2,3)-正規覆面算言語 $R_{2,3}^{regular}$ を受理するオートマトンである．黒丸が開始状態であり，四角が受理状態である．状態遷移を表す線の種類の違いは記号の種類の違いである

ン生成手順を，具体的に純であるかそうでないかの 2 通りで示し，実験では実際に生成されるオートマトンの状態数を調べた．

状態数の見積もりや実験からもわかる通り，進数 k が増えるにつれ生成されるオートマトンの状態数は爆発的に増加する．このことから，$k = 10$ の場合については明示的に決定性オートマトンを構築することは現実的ではないとわかった．ただし，k が小さい場合には実際に生成でき，数学的に解析することが可能であることもわかった．

今後は生成されたオートマトンを用いて受理する系列への番号付の方法を確立し，覆面算の番号付を行えるようにしたい．

参 考 文 献

1) H.E.Dudeney. *The Strand Magazine*, Vol. 68 of *Early British periodicals*. G. Newnes, 1924.
2) Joseph S. Madachy. *Madachy's Mathematical Recreations*. Dover, 1979.
3) 戸神星也, 渡辺治. 数独の解生成と解に対する番号付け, 2006.
4) D.E. Knuth. The Art of Computer Programming 4〔日本語版〕: Fascicle 2: Generating All Tuples and Permutations, 第 4 巻. アスキー・メディアワークス, 2006.
5) David Eppstei. *On the NP-Completeness of Cryptarithms*. Computer Science Department. 2000.