情報処理学会論文誌　データベース　Vol. 1　No. 2　1–14 (Sep. 2008)

ダブル配列を用いたAC法の効率的なパターン照合

蔵満琢麻[†1]　松浦寛生[†1]　望月久稔[†1]

パターン照合は文書処理やアンチウイルスなどのソフトウエアに用いられ，メモリ消費量が小さく，照合速度が高速なアルゴリズムが求められる．AC 法は複数パターンの照合に有効な手法で，AC マシンと呼ばれる一種の有限オートマトンを登録パターン集合から構築し，対象データを線形時間で照合する手法である．本論文では，ダブル配列を用いて遷移先関数を拡張した AC マシンを提案し，他手法との比較実験によりその有効性を示す．また提案マシンの応用例として，アンチウイルスソフト ClamAntiVirus に提案マシンを実装する．実験の結果，提案マシンは他手法よりも小さい記憶領域でデータ構造を実現し，対象データを高速に照合した．また，提案マシンを実装した ClamAntiVirus は，システムの稼働時間を 72%，照合時に必要な記憶領域を 70%にできることを示した．

Efficient Pattern Matching of Aho-Corasick Algorithm Using Double-Array

Takuma Kuramitsu,[†1] Nobutaka Matsuura[†1]
and Hisatoshi Mochizuki[†1]

Pattern matching is used for word processing and software such as antivirus. It is important to high-speed response and compact memory. Aho-Corasick algorithm is an efficient multiple pattern matching algorithm. In this paper, we present a multiple pattern matching machine with a double-array structure. It has the transition function extended. And also, we implement the proposal machine to ClamAntiVirus as an applied example. Our experiments show that the operation time decreased to 72% and required storage area decreased to 70%.

†1 大阪教育大学
Osaka Kyoiku University

1. はじめに

パターン照合は，コンピュータにおいて基本的な処理であり様々な場面で用いられるため，メモリ消費量が小さく，高速に照合するアルゴリズムが求められる．単一パターンを高速に照合するアルゴリズムとして BM 法[1] が有名だが，複数パターンを照合する場合，パターン数に応じて線形的に照合時間が増加する[2]．

ここで，複数パターンを対象データから高速に照合するアルゴリズム[3)–6)] の 1 つとして，Aho らにより提案された AC 法[3] があげられる．AC 法は，AC マシンと呼ばれる一種の有限オートマトンを登録パターン集合から構築し，対象データを線形時間で照合する手法であり，これまでに AC マシンの効率的な実現方法が研究されてきた．

有川ら[7] は，パターンの構成要素を分割して取り扱うことで各ノードにおける記憶領域を抑制し，決定性有限オートマトンに変換した AC マシンを提案した．この手法はデータベースから任意の情報を取り出すシステム[8] にも利用されている．このマシンは，パターンの構成要素を細かく分割するほど各ノードにおける遷移先を管理するための配列サイズを抑制できるが，照合時の遷移回数が増加するため，照合速度と照合時に必要な記憶領域がトレードオフの関係となる[7]．また，近年では，信種ら[9] により，ダブル配列[10),11)] を用いて高速性とコンパクト性をあわせ持つ AC マシンを実現する手法が提案されている．しかし，このマシンは，ダブル配列の遷移定義式により同一ノードへ複数の遷移を定義することができず，決定性有限オートマトンに変換できない．

本論文では，ダブル配列を用いた AC マシンに擬似ノードを付加して照合速度を高速化する方法を提案し，他手法との比較実験により提案マシンの有効性を示す．また，提案マシンの応用例として，検出率の高さから高い評価[12] を得ているアンチウイルスソフト ClamAntiVirus[13] に提案マシンを実装し，システムのパフォーマンスが向上することを示す．

以下，2 章では AC 法について簡単に説明し，従来手法について説明する．3 章で提案マシンを解説し，4 章で ClamAV への実装について述べる．5 章で提案手法を評価し，6 章でまとめと今後の課題を述べる．

2. AC 法

本章では AC マシンの構成とパターンの照合方法について説明する．次に遷移種を分割して登録した AC マシン[7] と，ダブル配列による AC マシン[9] の実現方法について説明する．

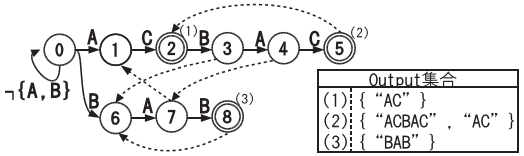


図 1　パターン集合 P を登録した AC マシン
Fig. 1　Aho-Corasick machine for the set of patterns P.

2.1　AC マシン

AC 法[3] は一種の有限オートマトンである AC マシンを登録パターン集合から構築して対象データを照合する．AC マシンは Goto 関数，Failure 関数，Output 関数から構成される．

Goto 関数は遷移種 a によるノード s からの遷移先ノード t を返す関数であり，Goto(s, a) $= t$ と表す．遷移が未定義である場合は失敗を表す fail を返す．Failure 関数は Goto 関数が fail を返した際に呼び出される関数で，遷移失敗時におけるノード s からの遷移先ノード f を返し，Failure(s) $= f$ と表す．以下，Goto 関数，Failure 関数による遷移をそれぞれ Goto 遷移，Failure 遷移と呼ぶ．

Goto 遷移には登録パターン集合によるトライ構造上の遷移と，初期ノードから初期ノードへの遷移が存在する．後者の遷移はトライ構造構築後，Goto(初期ノード，遷移種) が未定義であるときに作成される．

初期ノードからノード s までの遷移により構成されるパターンを ptn(s) と表記する．ノード s における Failure 遷移先は，ptn(f)（$f \neq s$）が ptn(s) の接尾辞となるノード f である．Failure 遷移先に該当するノードが複数存在する場合は，トライ構造上の深さが最も深いノードが選択され，また，1 つも存在しない場合は初期ノードが選択される．

Output 関数は，Goto 遷移により到達したノード s において検出するパターン集合 O_s を出力する関数で Output(s) $= O_s$ と表す．以下，Output 関数により出力するパターン集合 O_s を Output 集合と呼ぶ．

図 1 にパターン集合 $P = \{$“AC”, “ACBAC”, “BAB”$\}$ を登録した AC マシンを示す．AC マシン図における実線は Goto 遷移を表し，破線は初期ノード 1 以外への Failure 遷移を表す．また，2 重丸のノードは Output 集合を持つことを表し，ノード右上の括弧内における番号は Output 集合のインデクスを表す．たとえば図 1 において，Goto(0, ‘A’) $= 1$, Failure(4) $= 7$，Output(5) $= \{$“ACBAC”，“AC”$\}$ となる．

例 1：図 1 に示す AC マシンを用いて，照合対象データ $T =$ “$A_1C_2B_3A_4B_5C_6$” を照合する例を示す．対象データ T において，‘A’，‘B’，‘C’ はそれぞれ遷移種を表し，遷移種の添字は対象データ T における位置を表す．

まず初期ノード 0 において，Goto(0, ‘A_1’) $= 1$ より，ノード 1 へ Goto 遷移する．ノード 1 は Output 集合を持たないノードであるため，パターンを検出しない．次に Goto(1, ‘C_2’) $= 2$ よりノード 2 へ Goto 遷移する．ここで，ノード 2 は Output 集合を持つノードであるため，インデクス 1 に該当する Output 集合 $\{$“AC”$\}$ を検出する．同様に，遷移種 ‘B_3’，‘A_4’ により，ノード 3，4 と Goto 遷移する．ノード 4 において，Goto(4, ‘B_5’) が未定義であるため，Failure 関数を呼び出し，Failure(4) $= 7$ より，ノード 7 へ遷移する．Goto(7, ‘B_5’) $= 8$ よりノード 8 へ遷移し，Output(8) $= \{$“BAB”$\}$ を検出する．その後，Goto(8, ‘C_6’) $=$ fail，Failure(8) $= 6$ よりノード 6 へ Failure 遷移するが，ノード 6 においても Goto(6, ‘C_6’) $=$ fail となるため，初期ノード 0 へ Failure 遷移する．最後に Goto(0, ‘C_6’) $= 0$ より初期ノードから初期ノードへ Goto 遷移し，対象データの末尾まで照合したため照合を終える．

対象データ T を照合するために行った Goto 遷移回数は，対象データのサイズと等しい計 6 回，Failure 遷移回数は計 3 回，Output 集合の有無を判定した回数は計 6 回となる．（例終）

このように AC 法は，対象データを 1 度走査することで対象データに含まれるすべてのパターンを検出できる手法である．

しかし，Failure 遷移中は対象データの照合ポインタが進まないため，Failure 遷移の増加により AC マシンの照合速度は低下する．この問題は，各ノードにおいて未定義である Goto 遷移をあらかじめ定義することで解決できる[3),7)]．たとえば例 1 においては，Goto(4, ‘B_5’)，Goto(8, ‘C_6’) が未定義であったため，それぞれ Failure 遷移を得てからノード 8，ノード 0 へ Goto 遷移した．AC マシンの構築時にあらかじめ Goto(4, ‘B’) $= 8$，Goto(8, ‘C’) $= 0$ を定義することで，遷移先が決定的となり照合速度を高速化できる．

2.2　遷移種の分割による記憶領域の抑制

AC マシンを実現する方法として，各ノードにおける遷移を配列構造を用いて実現する[7),13)] ことが考えられる．しかし，各ノードに遷移種数分の要素を持つ配列を設けると，ノード数の増加により照合時に必要な記憶領域が膨大になる．以下，遷移先を管理する配列を遷移先管理配列と呼ぶ．

照合時に必要な記憶領域を抑制する方法の 1 つとして，遷移種を分割する手法[7)] が有川らにより提案されている．以下，有川らにより提案された AC マシンをマシン A とする．

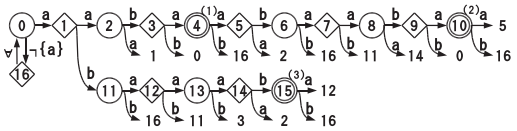


図 2　パターン集合 P を登録したマシン A
Fig. 2　Machine-A for the set of patterns P.

たとえば遷移種が 8 ビットで表現される場合，1 ノードあたりの遷移先管理配列サイズは 256 となる．ここで，遷移種を 4 ビットの 2 つに分割した場合，配列サイズは 16 となり，1 ノードあたりの遷移を定義するために必要な記憶領域は 16 分の 1 となる．1 つの遷移種を 2 つに分割することでノード数が 2 倍近くまで増加するが，遷移先管理配列に必要な記憶領域は 8 分の 1 以下となる[7]．

また，マシン A は Failure 遷移を除去して照合速度を高速化するため，マシン構築時にすべての Goto 遷移をあらかじめ定義する[7]．

パターン集合 P における遷移種を，A = aa，B = ab，C = ba とそれぞれ 2 分割するとき，パターン集合 P を分割して登録したマシン A を**図 2** に示す．図 2 において，丸で囲まれていない番号は遷移先のノード番号を表す．また，菱形で示すノードは遷移種の分割により生じる中間ノードであり，Output 集合を持たない．よって，照合時に Output 集合の有無を確認するのは 2 遷移に 1 回となる．

図 2 は遷移種を 2 分割して登録したため，照合対象のデータも同様に 2 分割する必要がある．図 2 を用いて，対象データ T を照合する例を以下に示す．

例 2：まず照合対象データ T の分割により，図 2 で照合するデータは “$a_1a_2b_3a_4a_5b_6a_7a_8a_9b_{10}b_{11}a_{12}$” となる．初期ノード 0 において，Goto(0, ‘a_1’) = 1 より，ノード 1 へ Goto 遷移する．同様に Goto(1, ‘a_2’) = 2 より，ノード 2 へ Goto 遷移する．ここで，2 回遷移したので Output 集合の有無を判定するが，Output(2) は存在しないため，ノード 2 において Output 集合を出力しない．同様に，遷移種 ‘b_3’，‘a_4’ により，順にノード 3，4 と Goto 遷移する．ここで，ノード 4 は Output 集合を持つため，Output(4)={“AC”} を出力する．次に遷移種 ‘a_5’，‘b_6’，‘a_7’，‘a_8’，‘a_9’，‘b_{10}’ により，順にノード 5，6，7，8，14，15 と Goto 遷移する．ここで，ノード 15 は Output 集合を持つため，Output(15) = {“BAB”} を出力する．その後，遷移種 ‘b_{11}’，‘a_{12}’ により，ノード 16，0 へ Goto 遷移し，対象データの末尾まで照合したので終了する．

対象データ T を照合するために行った Goto 遷移は計 12 回，Output 集合の有無を判定した回数は計 6 回となる．（例終）

マシン A は，決定性有限オートマトンに拡張したことにより照合時に Failure 遷移が発生しない．しかし，遷移種の分割により Goto 遷移回数が増加し，照合速度が低下する．例では遷移種を 2 分割したため，Goto 遷移回数が通常の AC マシンの 2 倍になった．

パターンの構成要素を細かく分割するほど各ノードにおける遷移先管理配列のサイズを抑制できるが，照合時の遷移回数が増加するため，照合速度と照合時に必要な記憶領域がトレードオフの関係となる[7]．

2.3　ダブル配列による AC マシン

AC マシンの一部分はトライ構造により実現される．トライ構造を効率的に実現する手法にダブル配列[10),11)] があり，ダブル配列を用いた AC マシン[9)]（以下，マシン B）が提案されている．

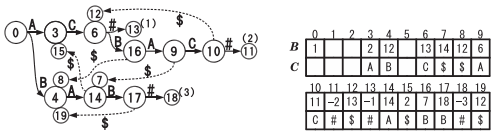
ダブル配列は配列 Base，Check を用いたトライ構造で，配列における添字はノード番号を表している．以下，ノード x における配列 Base，Check の要素を，それぞれ $B[x]$，$C[x]$ と表記する．ダブル配列は，ノード s から遷移種 a による遷移先ノード t を式 (1) により定義する[11)]．マシン B はトライ構造上の Goto 遷移を式 (1) を用いて定義する[9)]．

$$\begin{cases} t \leftarrow B[s] + a \\ C[t] = a \end{cases} \tag{1}$$

AC マシンの Goto 遷移には登録パターン集合によるトライ構造以外にも，初期ノードから初期ノードへ複数の遷移種による Goto 遷移が存在する．しかし，ダブル配列は遷移先の Check 値と遷移種が同種であることで遷移を確定するため，異なる遷移種による遷移を同一ノードへ定義できない．そこでマシン B は初期ノードから初期ノードへの Goto 遷移を，Goto(初期ノード，遷移種) が fail を返したときに対象データの照合ポインタを 1 つ進めることで実現する[9)]．

また，ダブル配列は，Base 値と遷移種の和を用いて遷移先のノード番号を決定するため，異なる Base 値を持つノードから同一ノードへの遷移を定義できない．

マシン B は初期ノード以外へ遷移する Failure 遷移を，新たなノードを作成して定義する．‘\$’ をパターンに使用しない特殊な遷移種として定義し，式 (2) で示すノード s_f の Base 値に，ノード s から Failure 関数により遷移すべきノード f の Base 値を写像することで Failure 関数を定義する．あるノード s で Failure 関数が呼び出されたとき，式 (2) により s から s_f への遷移の有無を判定し，遷移が存在すればノード s_f へ Failure 遷移し，存在し



	0	1	2	3	4	5	6	7	8	9
B	1			2	12		13	14	12	6
C				A	B		C	$	$	A

	10	11	12	13	14	15	16	17	18	19
	11	−2	13	−1	14	2	7	18	−3	12
	C	#	$	#	A	$	B	B	#	$

図 3　パターン集合 P を登録したマシン B
Fig. 3　Machine-B for the set of patterns P.

なければ初期ノードへ Failure 遷移する．

$$\begin{cases} s_f \leftarrow B[s] + \text{‘\$’} \\ C[s_f] = \text{‘\$’} \end{cases} \tag{2}$$

Failure 関数と同様に，Output 関数も新たなノードを作成して定義する．‘#’ をパターンに使用しない特殊な遷移種として定義する．式 (3) で示すノード s_o の Base 値に，ノード s における Output 集合のインデクス i を負値で格納することで Output 関数は定義される．照合時，Goto 遷移により遷移したノード s において，式 (3) により s_o の有無を判定し，s_o が存在すれば，s_o における Base 値の絶対値を参照して Output 集合を出力する．

$$\begin{cases} s_o \leftarrow B[s] + \text{‘\#’} \\ C[s_o] = \text{‘\#’} \end{cases} \tag{3}$$

図 3 にマシン B にパターン集合 P を登録したときのノード遷移図と配列 Base，Check に格納する値を示す．図 3 のマシン B において，遷移種 ‘#’，‘$’，‘A’，‘B’，‘C’ の内部表現値をそれぞれ 0，1，2，3，4 とする．また，Base 値，Check 値がともに空白のノードは未使用ノードであることを表す．‘#’，‘$’ による特殊遷移先のノードを小さい丸枠で示す．ノード 7 はノード 9 における Failure 遷移先のノードで，ノード 9 から Failure 遷移すべきノード 14 の遷移情報を持つ．同様に，ノード 15，8，19，12 は，それぞれノード 14，16，17，10 における Failure 遷移先を表し，ノード 3，4，4，6 の遷移情報を持つ．また，ノード 13，11，18 は，それぞれノード 6，10，17 における Output 集合のインデクスを管理する．図 3 を用いて対象データ T を照合する例を以下に示す．

例 3：まず初期ノード 0 において，式 (1) を用いて Goto 遷移の有無を確認する．$t = 3(B[0] + \text{‘A}_1\text{’})$，$C[3] = \text{‘A’}$ より，ノード 3 へ Goto 遷移する．ここで，式 (3) を用いてノード 3 における Output 集合の有無を判定する．$s_o = 2(B[3]+\text{‘\#’})$，$C[2] \neq \text{‘\#’}$ より，s_o への遷移が存在しないため Output 集合を出力しない．次に，遷移種 ‘C_2’ によりノード 6 へ Goto 遷移する．式 (3) より，$C[B[6]+\text{‘\#’}] = \text{‘\#’}$ であるため，$-B[B[6] + \text{‘\#’}] = 1$ に該当する Output 集合 {“AC”} を出力する．同様に，遷移種 ‘B_3’，‘A_4’ によりノード 16，9 と遷移する．ノード 9 において，遷移種 ‘B_5’ による Goto 遷移が未定義であり，ノード 9 が初期ノードではないため Failure 関数を呼び出す．式 (2) を用いて s_f の有無を確認する．$s_f = 7(B[9] + \text{‘\$’})$，$C[9] = \text{‘\$’}$ より，ノード 7 へ Failure 遷移する．次に，Goto(7, ‘B_5’) = 17 により，ノード 17 へ Goto 遷移する．このように，ノード 14 と等しい Base 値を持つノード 7 はノード 14 と同等の遷移情報を持つ．ノード 17 は Output 集合を持つため，$-B[B[17]+\text{‘\#’}] = 3$ に該当する Output 集合 {“BAB”} を出力する．Goto(17, ‘C_6’) は未定義であるため，ノード 19 へ Failure 遷移する．ノード 19 においても遷移種 ‘C_6’ による Goto 遷移が未定義であるため，再び Failure 関数を呼び出す．ここで，$C[13(B[19] + \text{‘\$’})] \neq \text{‘\$’}$ より，Failure 遷移が存在しないので初期ノード 0 に Failure 遷移する．初期ノード 0 においても，遷移種 ‘C’ による Goto 遷移は未定義であるため，対象データ T の照合ポインタを 1 つ進め，対象データの末尾まで照合したので終了する．

照合時に行った Goto 遷移回数は 6 回，Failure 遷移回数は 3 回となり，総遷移回数は 9 回となる．（例終）

マシン B は通常の AC マシンと同等の遷移回数で対象データを照合する．また，Output 集合や Failure 遷移をノードにより管理することで，マシン B の 1 ノードあたりに必要な記憶領域は Base 値と Check 値を保持する領域のみとなる．

しかし，異なる Base 値を持つノードから同一ノードへの遷移を定義できないため，マシン B は Goto 関数に決定性を持たせることができない．また，Failure 遷移する際にも ‘$’ 遷移の有無を確認する必要がある．

3. 遷移先関数を拡張したダブル配列による AC マシン

ダブル配列を用いた AC マシンにおける遷移回数を抑制し，照合速度を高速化する方法を提案する．本章では，まず，Output 集合の管理方法を変更して遷移情報を写像する方法について説明する．その後，擬似ノードを定義し，同一ノードへ複数の遷移を擬似的に作成して遷移回数を抑制する方法を説明する．最後に，拡張した遷移先関数について説明し，提案マシンの照合アルゴリズムを説明する．

3.1 Output 集合の管理方法

図 3 におけるノード 17 のように，Goto 遷移の存在しないノードにおいては，確実に

Function : Output(s)

(O1)　$s_o \leftarrow s$+‘#’;
(O2)　**if** ($C[s_o]$ =‘#’) **then**
(O3)　　**return** インデクス $B[s_o]$ の示す Output 集合;
(O4)　**return** empty;

図 4　関数：Output
Fig. 4　Function: Output.

Goto 遷移に失敗する．そのため，遷移先を確認するのは冗長である．ここで，Goto 遷移の存在しないノードに，あらかじめ Failure 遷移先の遷移情報を付加しておくことで，遷移先の決定性を高めることが考えられる．

ダブル配列は式 (1) を用いて遷移を定義するため，Base 値を写像することで遷移情報を写像できる[9]．しかし，マシン B は Goto 遷移の存在しないノードにおいても，Output 集合のインデクスを管理するノードへの遷移を持つ．そこで提案マシンは Output 集合の管理方法を変更し，ノード s における Base 値と，Output 集合のインデクスを管理するノード s_o との関連性を取り除く．

遷移種の最大値を a_{max} とし，‘#’ $= a_{max} + 1$ と定義する．式 (4) により定義するノード s_o の Base 値にノード s における Output 集合のインデクス i を格納することで Output 集合を管理する．以下，ノード s における Output 集合のインデクスを管理するノード s_o をノード s のレコードノードと呼ぶ．

$$\begin{cases} s_o \leftarrow s + \text{‘\#’} \\ C[s_o] = \text{‘\#’} \end{cases} \tag{4}$$

提案マシンにおける Output 関数を**図 4** に示す．Output 関数は，(O1)，(O2) でノード s における Output 集合のインデクスを管理するノード s_o の有無を判定する．s_o が存在すれば (O3) でインデクスに該当する Output 集合を返し，存在しなければ (O4) で Output 集合が存在しないことを示す empty を返す．

図 5 にノード s における Output 集合のインデクス i を管理するレコードノード s_o を四角で示す．図 5 において，ノード f は Failure 関数により遷移すべきノードを表し，ノード t はノード f から遷移種 a による Goto 遷移先を表す．

このようにレコードノードを定義することで，図 5 に示すように，Goto 遷移を持たないノード s の Base 値 $B[s]$ が未使用となる．この $B[s]$ に，Failure 遷移先 f の Base 値

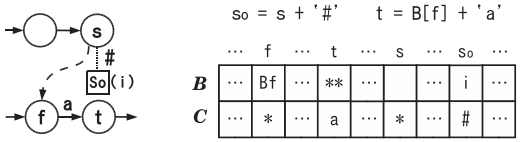


図 5　提案マシンにおける Output 集合の管理方法
Fig. 5　Output function of proposal machine.

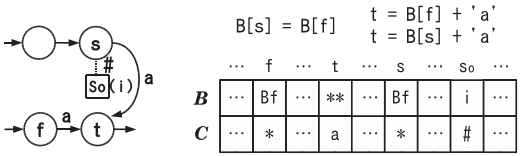


図 6　遷移情報の写像
Fig. 6　Copy of the transition.

$B[f] = B_f$ を写像する．図 5 におけるノード f の Base 値をノード s に写像したノード遷移図を**図 6** に示す．Base 値の写像によりノード s はノード f の遷移情報を持つ．このようにノード s から遷移種 a によるノード t への Goto 遷移を定義することで照合時の遷移回数を抑制できる．

3.2　擬似ノードによる遷移先の定義

各ノードにおいて，未定義の Goto 遷移をあらかじめ定義することで遷移回数を抑制することが考えられるが，ダブル配列は同一ノードへ複数の遷移を定義できない．そこで，既存ノード t への遷移を新たに作成する場合，ノード t と等価な情報を持つノード t_p を新たに作成し擬似的にノード t への遷移を定義する．以下，このとき作成するノード t_p を擬似ノードと呼ぶ．

作成した擬似ノード t_p には，ノード t の Base 値を写像する．また，ノード t が Output 集合を持つノードである場合は，t_p に同様の Output 集合を付加する．

図 7 に，擬似ノード t_p を用いてノード s から遷移種 ‘b’ による既存ノード t への遷移を擬似的に作成したノード遷移図を示す．図 7 において，擬似ノード t_p を三角で表す．擬似ノード t_p はノード t と等価な情報を持つ．このように擬似ノードを付加することで，異なる Base 値を持つノードから同一ノードへの遷移を定義できる．

提案マシンはこの擬似ノードを用いて初期ノードから初期ノードへの Goto 遷移を定義する．また，擬似ノードを用いて未定義である Goto 遷移を定義できるが，各ノードにおいて

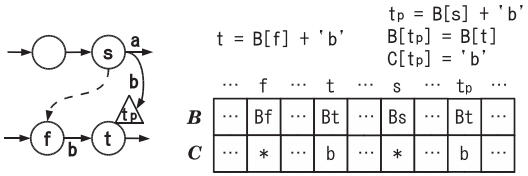


図 7　擬似ノード
Fig. 7　The pseudo-nodes.

すべての遷移種による遷移先を擬似ノードを用いて定義するとノード数が膨大になる．そこで，擬似ノードの付加条件を設ける．

3.3　擬似ノードの付加条件

ノード s における Check 値はノード s へ遷移するための遷移種を表す．つまり照合時において参照ノード s における Check 値は 1 つ前の遷移種を表す．ノード s において次に遷移すべきノード t のトライ構造上の深さが 2 以下であれば，ノード s における Check 値と対象データにおける遷移種を用いて遷移先を特定できる．そこで，遷移すべきノード t の深さが 2 以下となる場合には擬似ノードを作成しないという条件を設ける．

ノード s におけるトライ構造上の深さを Depth(s) とする．ただし，擬似ノードにおける深さを写像元のノードの深さと定義する．以下に示す条件 (1)，(2) をすべて満たす場合，提案マシンに擬似ノードを付加する．

(1)　Goto(s, a) = fail

(2)　Depth(遷移すべきノード t) ≥ 3

上記条件に該当する擬似ノードをすべて作成した後の AC マシンにおける Goto 関数について考える．あるノード s において遷移種 a による Goto 遷移が未定義であったとする．擬似ノード付加条件において作成する擬似ノードが，深さ 3 以上のノードへの Goto 遷移を保証するため，このときノード s から遷移種 a により Goto 遷移すべきノードは，トライ構造上の深さが 2 以下のノードに限定できる．

この性質を利用し，照合時に 1 つ前の遷移種に該当する $C[s]$ を用いて遷移先関数を拡張する．拡張した遷移先関数 NextMove を**図 8** に示す．NextMove 関数は，ノード s，遷移種 a を引数とし，ノード s から遷移種 a による遷移先 t を返す．(NM1)，(NM2) で式 (1) を用いて遷移の有無を判定する．式 (1) による遷移が未定義である場合，(NM3)〜(NM5) でトライ構造上の深さが 2 以下となるノードへの遷移を決定し，(NM7) で遷移先 t を返す．

拡張した遷移先関数 NextMove は必ず遷移先を返す．NextMove 関数を用いることで各ノードにおける遷移先の決定性が向上し，照合時間を短縮できる．

Function : NextMove(s, a)

```
(NM1)  t ← B[s] + a;
(NM2)  if (C[t] ≠ a) then
(NM3)      t ← B[B[ 初期ノード ] + C[s]] + a;
(NM4)      if (C[t] ≠ a) then
(NM5)          t ← B[ 初期ノード ] + a;
(NM6)  end
(NM7)  return t;
```

図 8　関数：NextMove
Fig. 8　Function: NextMove.

入力：対象データ $X = x_1x_2 \ldots x_n$　($x_i \in$ 遷移種)
出力：検出したパターンと対象データにおける位置

```
(M1)    s ← 初期ノード 0;
(M2)    for i ← 1 until n do
(M3)        s ←NextMove(s, x_i);
(M4)        O_s ←Output(s);
(M5)        if (O_s ≠ empty) then
(M6)            print O_s
(M7)            print i
(M8)        end
(M9)    end
(M10)   return
```

図 9　照合アルゴリズム
Fig. 9　Algorithm: Maching.

3.4　照合アルゴリズム

提案マシンは図 8 に示す NextMove 関数と図 4 に示す Output 関数を用いて**図 9** に示すアルゴリズムで対象データ $X = x_1x_2 \ldots x_n$ を照合し，検出したパターンと対象データにおける位置を出力する．対象データの長さを n とすると，提案マシンは (M3)〜(M7) に示す処理を n 回行うことで，対象データに含まれるパターンと対象データにおける位置を検出できる．

パターン集合 P を登録した提案マシンを**図 10** に示す．提案マシンにおいて，遷移種 'A'，'B'，'C'，'#' の内部表現値をそれぞれ 0，1，2，3 とする．レコードノードを四角枠で表し，擬似ノードを三角枠で表す．図 10 の提案マシンにおいて，ノード 3 とノード 8 が擬似

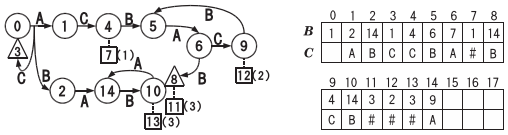


図 10　パターン集合 P を登録した提案マシン
Fig. 10　Proposal machine for the set of patterns P.

ノードに該当し，それぞれ初期ノード 0，ノード 10 における遷移情報を持つ．トライ構造構築後，Goto 遷移が存在しないノード 9，ノード 10 は，それぞれ Failure 遷移先に該当するノード 4，ノード 2 の遷移情報を持つので，ノード 5，ノード 14 への Goto 遷移が可能である．また，ノード 11 は擬似ノード 8 における Output 集合を管理するノードで，擬似ノード 8 とともに作成する．提案マシンを用いて，対象データ T を照合する例を以下に示す．

例　4：まず初期ノード 0 において，(M3) より NextMove(0, ‘A_1’) を呼び出す．$t = 1(B[0] + \text{‘}A_1\text{’})$，$C[1] =$ ‘A’ より遷移先はノード 1 となる．次に (M4) より Output(1) を呼び出す．(O1) より $s_o = 4(1 + \text{‘\#’})$，$C[4] \neq$ ‘#’ であるため，ノード 1 において Output 集合を検出しない．同様に，NextMove(1, ‘C_2’) = 4 よりノード 4 へ遷移し Output(4) を呼び出す．$s_o = 7(4 + \text{‘\#’})$，$C[7] =$ ‘#’ より，$B[7] = 1$ に該当する Output 集合 {“AC”} を返す．(M6)，(M7) で，検出した Output 集合 {“AC”} と，対象データにおける位置 2 を出力する．以下，同様に遷移種 ‘B_3’，‘A_4’，‘B_5’ により，順にノード 5，6，8 と遷移し，Output(8) = {“BAB”} より，(M6)，(M7) で，検出した Output 集合 {“BAB”} と，対象データにおける位置 5 を出力する．次に，(M3) より NextMove(8, ‘C_6’) を呼び出す．$t = 16(B[8] + \text{‘}C_6\text{’})$，$C[16] \neq$ ‘C’ より，(NM3)，(NM4) で遷移先候補を決め直す．$t = 16(B[B[0] + C[8]] + \text{‘}C_6\text{’})$，$C[16] \neq$ ‘C’ より，(NM5) で遷移先 t は $3(B[0] + \text{‘}C_6\text{’})$ となる．Output(3) = empty であり，対象データを末尾まで照合したため終了する．

提案マシンは Failure 遷移を行わないため，照合時の総遷移回数は対象データのサイズと等しく 6 回となる．（例終）

4. アンチウイルスソフトへの応用

提案マシンの応用例としてアンチウイルスソフト ClamAntiVirus ver0.92.1 に提案マシンを実装する．本章では，まず ClamAntiVirus の概要，照合部のデータ構造，照合方法について説明する．その後，提案マシンを実装して拡張したシステムについて説明する．

4.1 ClamAntiVirus

ClamAntiVirus（以下，ClamAV）はウイルスパターンを登録し，対象データと登録したウイルスパターンを照合してウイルスを検出するソフトであり，照合部に AC 法とハッシュ法を用いる[13]．

ClamAV に登録するウイルスパターンは部分的に不定なパターンを表すワイルドカードを含む．ClamAV はワイルドカードを含むウイルスパターンを AC 法を用いて処理し，含まないパターンをハッシュ法を用いて処理する．

しかし，AC マシンは不定なパターンを登録できないため，ClamAV はウイルスパターンの不定ではない部分をウイルスパターンのキーとして AC マシンに登録し，ウイルスパターン全体をキー検出時に出力する Output 集合として登録する．ClamAV は，パターンキー検出時に候補となるウイルスパターンと対象データを 1 バイトずつ照合することでワイルドカードを処理する．以下，キー検出時に照合するパターンを候補パターンと呼ぶ．

4.1.1 照合部データ構造

ClamAV はパターンキーを AC マシンに登録して，候補パターンを管理する構造体へのリンクを，キーを検出するノードに付加する．また，ワイルドカードを処理するために候補パターンを 1 要素 2 バイトの配列により管理する．1 バイトにパターンの構成要素を格納し，残りの 1 バイトにワイルドカードに対応したフラグを格納する．また，ハッシュテーブルに登録するウイルスパターンを 1 要素 1 バイトの配列を用いて管理する．

ClamAV における AC マシン（以下，マシン C）は配列構造を用いて Goto 遷移を定義する．マシン C の各ノードは，遷移先管理配列へのリンク，Failure 関数による遷移先へのリンク，Output 集合を管理する構造体へのリンク，遷移先管理配列の有無を判定するフラグ，Output 集合の有無を判定するフラグの 5 つの要素により構成される．

マシン C は Goto 遷移を持つ各ノードに遷移種数分のサイズを持つ遷移先管理配列が必要となるため，ノード数の増加により照合時に必要な記憶領域が膨大になる．そこで照合時の記憶領域を抑制するため，マシン C に登録するウイルスパターンのキーを 3 バイト以下に制限する．

ClamAV は制限したキー長以下の不定ではない部分パターンのうち，最長のものをパターンキーとして AC マシンに登録する[13]．

次にハッシュテーブルについて説明する．ClamAV に用いられるハッシュ関数を式 (5) に示す．ClamAV はパターンキーを式 (5) によりハッシングして，候補パターンをハッシュテーブルに登録する．式 (5) において，p はパターンキーの先頭を表し，$in[p]$ はパターン

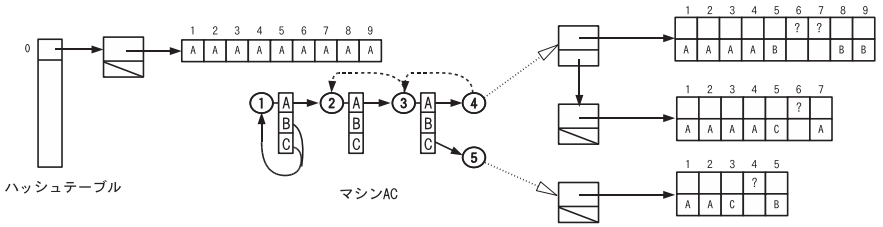


図 11　ClamAV のデータ構造
Fig. 11　The data structure of ClamAV.

キーの先頭要素を表す．

$$211 * in[p] + 37 * in[p+1] + in[p+2] \quad (5)$$

ハッシュテーブルに新たに登録するパターンにおいて，ClamAV は登録済みのパターンキーとハッシュ値が重複しない部分パターンをパターンキーとして選択する．該当する部分パターンが存在しない場合はパターンの末尾 3 バイトをパターンキーとする[13]．

図 11 にパターン集合 Q = {“AAAAAAAAA”，“AAAAB??BB”，“AAAAC?A”，“AAC?B”} を登録した ClamAV におけるウイルスデータベースのデータ構造を示す．ここで，パターン内の ‘?’ は任意の 1 バイトを示すワイルドカードを表し，照合時においてすべての遷移種と一致する．また，パターン “AAAAB??BB”，“AAAAC?A”，“AAC?B” において AC マシンに登録するキーは，それぞれの先頭 3 バイト “AAA”，“AAA”，“AAC” となり，ハッシュテーブルに登録するパターン “AAAAAAAAA” のキーは “AAA”（ハッシュ値 0）となる．

4.1.2　照 合 方 法

ClamAV はまずハッシュテーブルに登録した候補パターンと対象データを照合し，その後，AC 法により対象データを照合する．また，照合中にウイルスパターンを検出した場合は，その時点で照合を終える．

まずハッシュ法による照合方法を説明する．照合時，式 (5) における p は照合対象データの照合ポインタを表し，$in[p]$ は照合対象データにおける照合ポインタが示す位置の要素を表す．対象データの照合ポインタを 1 つずつ進めながら候補パターンを照合し，$p+1$ が照合対象データの末尾を指した時点でハッシュ法による照合を終える．

次に AC 法による照合方法を説明する．ClamAV は照合対象データを AC マシンに入力し，パターンキーを検出するまで対象データの照合ポインタを進める．キーを検出した際，該当する候補パターンを 1 つずつ取り出し，キー以外の部分を対象データと照合する．

図 11 を用いて対象データ U = “$A_1A_2A_3A_4B_5C_6A_7B_8C_9A_{10}A_{11}A_{12}A_{13}C_{14}A_{15}B_{16}$” を照合する例を以下に示す．例における遷移種 ‘A’，‘B’，‘C’ の内部表現値をそれぞれ 0，1，2 とする．

例 5：まず，対象データ U をハッシュ法により照合する．対象データ U の照合ポインタ p を 1 に設定し，$in[p]$ = ‘A_1’，$in[p+1]$ = ‘A_2’，$in[p+2]$ = ‘A_3’ より式 (5) を用いてハッシュ値 0 を求め，ハッシュテーブルに登録した候補パターン “AAAAAAAAA” と対象データを 1 バイトずつ照合する．ここで，候補パターンの 5 つ目の要素 ‘A’ により照合に失敗するため，p の値を 1 つ増やして再び対象データ U をハッシングする．以下，同様の動作を繰り返す．この例では，p の値が 1，2，10，11 のときにハッシュテーブルに登録した候補パターンと対象データを照合するがすべて失敗する．

次に再び p を 1 に設定し，対象データ U を AC マシンに入力する．遷移種 ‘A_1’，‘A_2’，‘A_3’ により初期ノード 1 からノード 2，3，4 へ遷移し，p = 3 のときに Output(4) = {“AAAAB??BB”，“AAAAC?A”} を得る．ここで，対象データ U の p 以降を，Output(4) により得た候補パターンと 1 つずつ照合する．しかし，それぞれ候補パターンの 9 つ目の要素 ‘B’ と 5 つ目の要素 ‘C’ により照合に失敗する．同様に，AC マシンによる照合を進めると，p の値が 4，12，13 のときにそれぞれ Output(4) を得るが，候補パターンの照合にすべて失敗する．その後，$p = 14$ のとき，Output(5) = {“AAC?B”} を得る．候補パターンを U と照合すると一致するため，ウイルスパターン “AAC?B” を検出して終了する．

対象データ U を照合するために要した候補パターンの照合回数は 13 回，また，Failure 遷移を含めた AC マシンにおける総遷移回数は 21 回となる．（例終）

ClamAV はハッシュ法と AC 法との併用により対象データを 2 回走査する必要がある．また，パターンのキー長を 3 バイトに制限することによりキーが重複し，照合時に複数の候補パターンが検出される．候補パターンの照合中は対象データの照合ポインタは進まないため，候補パターンの照合回数が増加すると照合速度が低下する．

4.2　提案システム

マシン C が 1 ノードあたりに遷移種数分のサイズを持つ遷移先管理配列が必要であるのに対し，提案マシンは Base 値，Check 値を保持する領域だけでよく，1 ノードあたりに必要な記憶領域が小さいため，同様のパターン集合を登録した場合，照合時に必要な記憶領域が小さい．そこで，すべてのウイルスパターンのキーを提案マシンに登録し，AC 法のみにより照合するシステムを提案する．

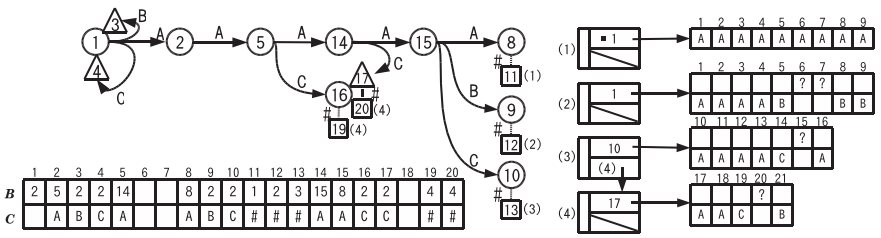


図 12　提案システムのデータ構造
Fig. 12　The data structure of proposal system.

さらに，提案マシンにウイルスパターンのキー長を伸ばして登録し，Output 集合を持つノードを分散させることで候補パターン照合回数の抑制を図る．

4.2.1　照合部データ構造

提案システムにおいて，ワイルドカードを含むパターンに対しては ClamAV と同様の方法でパターンキーを決定し，含まないパターンに対しては，制限したキー長以下の接頭辞をパターンキーとして AC マシンに登録する．

図 12 にパターン集合 Q を登録した提案システムのデータ構造を示す．AC マシンに登録するウイルスパターンのキー長を 5 とし，遷移種 ‘A’，‘B’，‘C’，‘#’ の内部表現値をそれぞれ 0，1，2，3 とする．

提案システムは，候補パターンへのリンクをカーソル管理により実現する．ここで，ワイルドカードを含まない候補パターンと含む候補パターンとでは管理する配列が異なるため，カーソルを管理する変数の最上位ビットをフラグとして用いる．ノード 8 において検出する候補パターンはワイルドカードを含まないため，候補パターンへのリンクを管理する変数のフラグを立てる．図 12 の中点はフラグが立っていることを表す．

4.2.2　照 合 方 法

AC 法によりウイルスパターンのキーを検出し，対象データとキーに該当する候補パターンを照合する．また，ワイルドカードを ClamAV と同様の方法で処理する[13]．図 12 に示すウイルスデータベースを用いて入力パターン U を照合する例を以下に示す．

例 6：まず，対象データ U の照合ポインタ p を 1 に設定し，提案マシンに U を入力する．遷移種 ‘A_1’，‘A_2’，‘A_3’，‘A_4’，‘B_5’ により，初期ノード 1 からノード 2，5，14，15，9 へ遷移し，$p = 5$ のとき，Output(9) = {“AAAAB??BB”} を得る．候補パターンを対象データ U と照合するが，最後の要素 ‘B’ で失敗する．同様に提案マシンによる照合を進めると，$p = 14$ のとき，Output(10) = {“AAAAC?A”，“AAC?B”} を得る．1 つ目の候補パターン “AAAAC?A” と対象データ U との照合に失敗するが，2 つ目の候補パターン “AAC?B” との照合に成功するため，ウイルスパターン “AAC?B” を検出して終了する．候補パターンの照合回数は 3 回であり，提案マシンにおける総遷移回数は 14 回となる．（例終）

例 5 と比較すると，AC マシンにおける総遷移回数，候補パターン照合回数が少なく，提案システムのほうが効率良く照合できることが分かる．

5.　評　　価

本章では，他手法との比較により提案マシンの照合時間と照合時に必要な記憶領域について評価した後，提案システムと ClamAV を実験により評価する．

5.1　提案マシンの評価

パターンを分割して登録したマシン A[7]，ダブル配列によるマシン B[9]．ClamAV ver0.92.1[13] に実装されているマシン C を比較手法とし，提案マシンを理論的・実験的に評価する．ただし，パターンの構成要素を 1 バイトとし，マシン A にはパターンを 4 ビットに分割して登録する．

まず照合時間について評価する．照合時間を遷移回数と出力判定の和と考え，対象データのサイズを m とする．提案マシンは遷移先関数の拡張により照合時に Failure 遷移が発生しないため，総遷移回数は Goto 遷移回数 m と等しい．マシン B，マシン C は Failure 遷移が最小で 0 回，最大で $m-1$ 回発生する．Goto 遷移回数は提案マシンと等しく m 回であるため，総遷移回数は m〜$(2m-1)$ 回となる．マシン A は決定性有限オートマトンへの拡張により Failure 遷移が発生しない．しかし遷移種を 4 ビットに分割するため Goto 遷移回数が $2m$ 回となり，総遷移回数は $2m$ 回となる．また，各手法とも出力判定の回数は m 回である．よって，提案マシン，マシン A，マシン B，マシン C における照合時間はそれぞれ，$2m$，$3m$，$2m$〜$(3m-1)$，$2m$〜$(3m-1)$ となり，提案マシンは高速であるといえる．

次に照合時に必要な記憶領域について評価する．記憶領域を 1 ノードあたりに必要な記憶領域とノード数との積として考える．レコード情報の管理領域は，要件によって変化するためここでは考えない．

提案マシン，マシン B における 1 ノードあたりに必要な記憶領域は，Base 値と Check 値のみである．Base 値と Check 値を 4 バイトの変数にまとめて格納する．Check 値に必要な情報量は，‘#’ や ‘$’ などの特殊遷移を含めると 9 ビットであるため，残りの 23 ビットを Base 値として用いる．

ただし，図 3 に示すように，ダブル配列上には未使用領域が存在するため，マシン構築時における初期配列の大きさは大きめに確保する必要がある．本実験においては，初期配列の大きさを 23 ビットの Base 値で表現できる 800 万とした．

マシン A は，パターンの構成要素を 4 ビットとして取り扱うため，遷移先管理配列の配列サイズは 16 となる．マシン A における遷移先管理配列を 1 要素 3 バイトの配列を用いて，カーソル管理によりノード間のリンク構造を実現すると，遷移先管理配列に必要な記憶領域は 48 バイトとなる．さらに Output 集合のインデクスを管理する領域が 4 バイト必要となるため，1 ノードあたりの記憶領域は 52 バイトとなる．

マシン C はポインタ管理によりリンク構造を実現するため，1 要素 4 バイトの遷移先管理配列を用いる．マシン C の各ノードは，遷移先管理配列へのリンク，Failure 関数による遷移先へのリンク，Output 集合を管理する構造体へのリンク，遷移先管理配列の有無を判定するフラグ，Output 集合の有無を判定するフラグの 5 つの要素により構成される．ClamAV は各リンク情報を 4 バイトの変数を用いて管理し，フラグを 1 バイトの変数を用いて管理する．また，パターンの構成要素を 1 バイトずつ取り扱うため，遷移先管理配列の配列サイズは 256 となる．これにより 1 ノードあたりに必要な記憶領域は 1,038 バイト（遷移先管理配列が存在しないノードは 14 バイト）となる．

ここで，マシン C と同様のアルゴリズムで，1 要素 3 バイトの配列を用いて，カーソル管理によりリンク構造を実現した AC マシン（以下，マシン C+）を新たに作成した．以下，マシン C+もあわせて評価する．マシン C のアルゴリズムにおいて，リンクの有無を確認するためにフラグは必要ないと考えられるため，作成したマシン C+においてはフラグの管理領域を削除した．これにより，1 ノードあたりに必要な記憶領域は 780 バイト（遷移先管理配列が存在しないノードは 12 バイト）となる．

1 ノードあたりに必要な記憶領域は提案マシンがマシン B と同様に最も小さいが，擬似ノードやレコードノードの付加によりノード数が増加するため，照合時に必要な記憶領域は登録パターン集合に依存する．そこで，提案マシンに付加する擬似ノードの数について評価する．擬似ノードは，遷移先を定義するために付加される．よって各ノードに付加する擬似ノード数は最高で遷移種数となる．これは最悪の場合，提案マシンの記憶領域が配列構造により遷移先を管理した AC マシンとほぼ同等となることを示す．

ここで，擬似ノードがどのように増加するか考察する．初期ノードから初期ノードへの遷移先として付加する擬似ノード数はたかだか遷移種数となるため，問題にはならない．そこで初期ノード以外への遷移先として付加する擬似ノードについて考える．

擬似ノードの付加条件により，擬似ノードを付加するノードは，Failure 遷移先のノードの深さが 2 以上となるノードである．ここで，ノード s における Failure 遷移先は，$\mathrm{ptn}(f)$（$f \neq s$）が $\mathrm{ptn}(s)$ の接尾辞となるノード f と定義される．登録パターン数が増加するほど，初期ノードから到達できるノード数も増加するため，Failure 遷移先のトライ構造上の深さが深くなる可能性が高い．よって，登録パターン数の増加にともない，擬似ノード数の増加率が上昇するといえる．

初期ノードから遷移種 $a_1, a_2, \ldots, a_n$（$n \geq 3$）により到達するノード s_n において，$\mathrm{Goto}(s_n,\ a_{n+1})$ が未定義であるとき，$\mathrm{Goto}(s_n,\ a_{n+1})$ を定義するために擬似ノードを付加する確率 $p(n)$ を考える．

$p(n)$ は，ノード s_n からの Failure 遷移先となるノード f の深さが 2 以上で，$\mathrm{Goto}(f,\ a_{n+1})$ が定義されている確率と等価である．よって $p(n)$ は，パターン $a_2 a_3 \ldots a_n a_{n+1}$ の接尾辞のうちで長さが 3 以上となるパターンにより初期ノードからたどり着くノードが存在する確率となる．

ここで，登録パターン集合において，すべての遷移種が一様に分布すると仮定すると，初期ノードから任意の遷移種 $b_1, b_2, \ldots, b_m$ により到達するノード s_m が存在する確率 $q(m)$ は式 (6) で表すことができる．

$$q(m) = \frac{\min\left(\text{パターン数},\ \text{遷移種数}^m\right)}{\text{遷移種数}^m} \tag{6}$$

式 (6) において遷移種数 ^{m} は m 個の遷移種がとりうる全組合せの数を表す．また，min(パターン数，遷移種数 ^{m}) は，パターン集合登録後のトライ構造において，深さが m となるノードの数を表し，パターン数と遷移種数 ^{m} のうち小さい方の値をとる．式 (6) を用いると，$p(n)$ は式 (7) で表すことができる．

$$p(n) = \min\left(\sum_{k=3}^{n} q(k),\ 1\right) \tag{7}$$

パターン数 $\ll$ 遷移種数 ^{3} であるとき，$p(n)$ を $q(3)$ で近似すると，擬似ノードの付加条件 (2) は，深さが 3 以上のノードに付加する擬似ノード数を，条件を設けない場合の $\left(\text{パターン数} \,/\, \text{遷移種数}^3\right)$ 倍に抑制する条件といえる．しかし，実際にマシンに登録するパターン集合の遷移種が一様に分布するとは限らないため，提案マシンにおける記憶領域は実験により詳細に評価する必要がある．

提案マシンを評価するために，他手法との比較実験を Intel Pentium Dual 1.60 GHz,

表 1　登録パターン数 5 千個，8 万個における実験結果
Table 1　The results for 5,000 and 80,000 patterns.

	総遷移回数		ノード数		照合時間 (s)		記憶領域 (byte)	
	5 千個	8 万個	5 千個	8 万個	5 千個	8 万個	5 千個	8 万個
提案マシン	20,000,000	20,000,000	32,280	889,112	0.64	2.07	129,120	3,556,448
マシン A	40,000,000	40,000,000	30,595	403,443	1.22	3.07	1,590,992	20,979,088
マシン B	38,600,388	33,108,595	43,710	601,916	0.91	3.04	174,840	2,407,664
マシン C	38,600,388	33,108,595	16,078	219,013	1.51	5.01	12,299,488	160,312,400
マシン C+	38,600,388	33,108,595	16,078	219,013	1.59	4.60	9,224,618	120,234,302

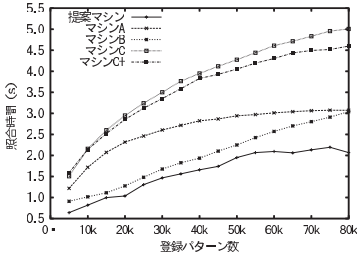


図 13　照合時間
Fig. 13　Matching time.

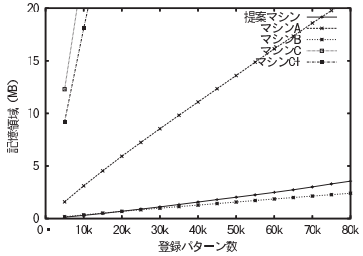


図 14　照合時に必要な記憶領域
Fig. 14　Memory required for matching.

Fedora 7 上で行った．実験では，2008 年 3 月 1 日時点のウイルスパターン[13] から，2 バイト以上，5 バイト以下の，実際にシステムに登録するパターンキーを抽出して 8 万個のパターンを作成し，作成したパターン集合（以下パターン集合 R とする）を母集団として用いた．また，パターン集合 R を母集団として 5 千個から 8 万個まで 5 千個ごとにパターン数を増やしたパターン集合を作成し，パターン集合 R を連結して作成した 20 MB のファイルを照合対象ファイルとして各パターン集合と照合した．

それぞれの実験における照合時間を**図 13** に，照合時に必要な記憶領域を**図 14** に示す．また，**表 1** に，5 千個のパターン，8 万個のパターンを各手法に登録したときの照合時における総遷移回数，ノード数，照合時間，照合時に必要な記憶領域を示す．ただし，ダブル配列を用いる提案マシン，マシン B におけるノード数は，ダブル配列上の未使用領域を含む．以下，ダブル配列上の未使用領域を空きノードと呼び，レコードノード，擬似ノード，空きノードを除くノードを一般ノードと呼ぶ．**表 2** には提案マシンの一般ノードにおける Failure 遷移先（表中，F 遷移先と表記）の深さ平均とノード数の内訳を示す．

まず照合時間について評価する．図 13 に示すように，提案マシンは最も高速であることが分かる．これは，表 1 に示すように，遷移先関数を拡張したことにより提案マシンの総遷移回数が他手法よりも少ないためである．

次に記憶領域について，実験結果から評価する．表 2 に示すように，登録パターン数の増加により，提案マシンに付加する擬似ノード数の割合は増加する．これは Failure 遷移先のトライ構造上の深さが増し，擬似ノードを付加するノードが増加したためである．これにより，表 1 に示すように，8 万個のパターンを登録した場合，提案マシンのノード数は他手法より多い．

しかし，提案マシンは 1 ノードあたりの記憶領域がマシン B 同様最も小さいため，今回の実験においては図 14 に示すように，マシン A，C，C+と比べて遥かに小さい記憶領域で照合した．また，5 千個のパターンを登録した場合，表 1 に示すように，提案マシンの記憶領域はマシン B よりも小さかった．これは，擬似ノードの付加条件がある程度のパターン数まで特に有効に機能することを表す．

表 2　Failure 遷移先深さ平均とノード数内訳
Table 2　The average of depth of the node reached by the failure function and the details of the number of nodes.

	F 遷移先	ノード数				ノード数
	深さ平均	一般ノード数	空きノード数	レコードノード数	擬似ノード数	合計
5 千個	1.23	16,078 (49.8%)	7,512 (23.3%)	4,932 (15.3%)	3,758 (11.6%)	32,280
8 万個	1.68	219,013 (24.6%)	219,193 (24.7%)	88,264 (9.9%)	362,642 (40.8%)	889,112

() 内はノード数合計を 100%としたときの割合

表 3　システムにおけるパターンの最大キー長別の実験結果
Table 3　The results according to the max length of pattern-key of the systems.

	最大キー長	3	4	5	6	7	8	9	10	11	12
提案システム	稼働時間 (s)	13.33	10.78	10.00	9.73	9.53	9.81	9.97	9.77	9.71	9.65
	照合時間 (s)	9.77	7.20	6.43	6.15	5.95	6.22	6.34	6.19	6.12	6.07
	読み込み時間 (s)	3.45	3.46	3.46	3.46	3.46	3.47	3.51	3.46	3.47	3.46
	記憶領域 (MB)	14.37	15.19	16.42	17.89	19.48	21.16	22.91	24.59	26.46	28.23
	P 照合回数 (M 回)	121.56	72.05	55.97	50.49	47.94	49.61	51.06	50.87	47.16	47.07
	ノード数 (k 個)	210	415	722	1089	1487	1906	2344	2766	3233	3676
ClamAV	稼働時間 (s)	14.98	14.77	14.56	14.66	14.65	14.90	15.12	15.10	15.19	15.25
	照合時間 (s)	10.31	10.03	9.83	9.90	9.90	10.10	10.24	10.26	10.29	10.33
	読み込み時間 (s)	4.46	4.52	4.51	4.55	4.54	4.58	4.66	4.62	4.67	4.69
	領域 (MB)	21.58	27.27	33.66	40.58	47.97	55.66	63.35	70.99	78.58	86.13
	P 照合回数 (M 回)	63.51	56.83	55.34	55.03	54.93	56.85	58.53	58.50	58.46	58.44
	ノード数 (k 個)	10.04	16.27	22.99	30.04	37.43	45.11	52.70	60.22	67.62	74.96
C+システム	稼働時間 (s)	15.48	15.10	14.98	14.98	14.97	15.24	15.49	15.59	15.46	15.62
	照合時間 (s)	10.89	10.52	10.39	10.38	10.37	10.65	10.89	10.98	10.83	11.01
	読み込み時間 (s)	4.40	4.38	4.39	4.40	4.41	4.40	4.40	4.41	4.44	4.41
	領域 (MB)	20.26	24.51	29.28	34.45	39.97	45.72	51.46	57.17	62.84	68.48
	P 照合回数 (M 回)	63.51	56.83	55.34	55.03	54.93	56.85	58.53	58.50	58.46	58.44
	ノード数 (k 個)	10.04	16.27	22.99	30.04	37.43	45.11	52.70	60.22	67.62	74.96

5.2　提案システムの評価

ClamAV との比較実験により，システムの稼働時間や照合時間，ウイルスパターン照合時に必要な記憶領域について評価する．また，最大キー長の変動による候補パターン照合回数の増減について説明する．

提案システムと ClamAV(ver.0.92.1) との比較実験を Intel Pentium Dual 1.60 GHz，Fedora7 上で行った．実験では 2008 年 3 月 1 日時点のウイルスパターン[13] をシステムに登録し，ファイル数 21，総バイト数約 9.38 MB の EXE ファイルを照合した．AC マシンに登録するウイルスパターンの最大キー長を 3〜12 まで変動させ，それぞれ実験した．また，ClamAV における AC マシンをマシン C+に改良して記憶領域を抑制したシステム（以下，C+システム）も比較手法として用いた．

実験におけるシステムの稼働時間，ウイルスパターンの照合時間，ウイルスデータベースの読み込み時間，照合時に必要な記憶領域，照合時の候補パターン照合回数（表中，P 照合回数），AC マシンにおけるノード数を**表 3** に示す．ただし，表 3 に示す照合時間は候補パターンの照合時間を含み，記憶領域はウイルスパターンの管理領域を含む．

まず，ウイルスデータベースの読み込み時間について評価する．表 3 に示すように，提案システムにおけるウイルスデータベースの読み込み時間は最大キー長に依存せずに，最大キー長 3 における ClamAV の約 77%になる．これは，ClamAV がシステムを起動するたびにウイルスデータベースの構築が必要となるのに対し，提案システムが構築済みのウイル

スデータベースをファイルから直接復元できるためである．提案システムと同様にカーソル管理により AC マシンを構築する C+システムの読込時間が提案システムよりも長いのは，C+システムが AC マシンのほかにハッシュテーブルの構築も必要とするためである．

次に，最大キー長の変動による候補パターン照合回数の増減について説明する．

表 3 に示すように，今回の実験において，パターンの最大キー長を 3〜7 に制限した場合，キー長の上限が大きいほど候補パターンの照合回数が減少し，照合時間が減少する．これは，キー長を伸ばすことで Output 集合を持つノードが分散するためである．しかし，最大キー長の上限を 7 から 8 にした場合，候補パターン照合回数が増加する．これは，最大キー長が変動することでキーのパターンにおける始点が変化する可能性があるためである．たとえば，以下に示す 2 つのパターンを考える．

パターン 1： “ABC?AAAA?AAAAA”

パターン 2： “DEF?AAAA?AAAAB”

最大キー長を 3 に制限した場合，パターン 1，2 におけるキーはそれぞれ ABC，DEF となり，候補パターンは異なるノードが保持する．しかし，最大キー長が 4 である場合にはパターン 1，2 におけるキーはどちらも AAAA となるため，同一ノードが複数の候補パターンを保持する．また，それぞれのパターンキーの検出率が一定ではないため，検出される可能性の高いパターンキーが選択された場合，候補パターンの照合回数が増加する．

以上の理由により，候補パターン照合回数は最大キー長の増加による単調減少にはならない．キー長を伸ばすことで照合時の記憶領域が増加するため，環境に応じて適当なキー長を選択する必要がある．

実験結果から，照合時間について評価する．表 3 に示すように．今回の実験において，提案システムは最大キー長が 7 のときに最も高速に対象データを照合し，提案システムにおける照合時間は同様のキー長における ClamAV の約 60%となった．これは，提案システムが AC 法のみを用いて照合するうえに，候補パターンの照合回数が ClamAV よりも少ないためである．また，最大キー長 3 において，提案システムの候補パターン照合回数は ClamAV を大きく上回るが，照合時間は ClamAV と同程度であった．これは提案マシンの照合速度が ClamAV における AC マシンよりも高速であることを表す．

次に，照合時の記憶領域について評価する．提案システムはすべてのウイルスパターンのキーを AC マシンに登録するため，表 3 に示すように，ClamAV における AC マシンよりもノード数が多い．しかし，最大キー長 7 における提案システムの照合時記憶領域は，キー長 3 における C+システムの記憶領域よりも小さい．これは提案マシンの 1 ノードあたりの記憶領域がマシン C+よりもはるかに小さいためである．照合時間が ClamAV と同程度となる最大キー長 3 における提案システムの記憶領域は，同じキー長における ClamAV の約 67%であった．

最後に，稼働時間と記憶領域を総合的に評価するために，現状の ClamAV を基準とし，稼働時間の比率と記憶領域の比率との和を指標として考える．現状の ClamAV は最大キー長を 3 に制限されており，表 3 よりこの和が最小になる．これに対して，提案システムは最大キー長を 4 に制限したときに稼働時間の比率と記憶領域の比率との和が最小になる．このとき，提案システムは ClamAV に対し，記憶領域を約 70%に抑制し，稼働時間を約 72%に短縮した．

以上より，他手法との比較により提案マシンの有効性を示した．また，提案マシンをシステムに実装することで，照合時に必要な記憶領域を抑制し，照合時間を短縮できることを示した．

6. おわりに

本論文では，ダブル配列に擬似ノードを付加して遷移先関数を拡張した AC マシンを提案し，他手法との比較実験を行った．実験の結果，提案マシンは他手法よりも小さい記憶領域でデータ構造を実現し，対象データを高速に照合した．また，提案マシンの導入により拡張した ClamAV はシステムの稼働時間を 72%，照合時に必要な記憶領域を 70%に抑制できることを示した．提案マシンは，アンチウイルスソフトのほかにも，データをストリーム処理する分野に応用できると考えられる．

擬似ノードを付加することでダブル配列はトライ構造を越える表現能力を持つ．これによりダブル配列の応用範囲がさらに広がるものと思われる．今後の課題として，正規表現を含むパターンに対応させることがあげられる．

しかし，擬似ノードの付加により各ノードにおける出次数が増加する．ダブル配列を構築する際には，ノード番号が重複しないように各ノードの Base 値を決定する必要があるため[10),14)]，各ノードにおける出次数の増加により構築速度が低下する．そこで，提案マシンの効率的な構築法も今後の課題としてあげられる．

参考文献

1) Boyer, R.S. and Moore, J.S.: A Fast String Searching Algorithm, *Comm. ACM*, Vol.20, No.10, pp.762–772 (1977).

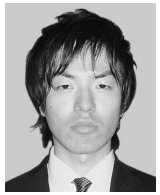
2) Aho, A.V.：文字列中のパターン照合のためのアルゴリズム，アルゴリズムと複雑さ，丸善，chapter5, pp.263–304 (1994).
3) Aho, A.V. and Corasick, M.J.: Efficient String Matching An Aid to Bibliographic Search, *Comm. ACM*, Vol.18, No.6, pp.333–340 (1975).
4) 浦谷則好：高速な複数文字列照合アルゴリズム：FAST，情報処理学会論文誌，Vol.30, No.9, pp.1119–1125 (1989).
5) 広瀬貞樹，小栗伸幸，吉澤寿夫，山淵龍夫：複数パターンの文字列照合におけるマッチングマシンの動的構成法，電子情報通信学会論文誌，Vol.J80-DI, No.8, pp.655–664 (1997).
6) Kärkkäinen, J. and Sanders, P.: Simple Linear Work Suffix Array Construction, *Proc. 30th International Conference on Automata, Languages and Programming*, Vol.2719, pp.943–955 (2003).
7) 有川節夫，篠原　武：文字列パターン照合アルゴリズム，コンピュータソフトウェア，Vol.4, No.2, pp.98–119 (1987).
8) 有川節夫，篠原　武，宮原哲浩，武谷峻一，宮野　悟，竹田正幸，大島一彦，白石修二，酒井　浩，山本章博：テキストデータベース管理システム SIGMA とその利用，情報処理学会研究報告，Vol.1989, No.1989-FI-014, pp.1–8 (1989).
9) 信種真人，森田和宏，泓田正雄，青江順一：ダブル配列を用いたマシン AC の効率的格納方法，言語処理学会第 13 回年次大会，pp.831–834 (2007).
10) Aoe, J.: An Efficient Digital Search Algorithm by Using a Double-Array Structure, *IEEE Trans. Softw. Eng.*, Vol.15, No.9, pp.1066–1077 (1989).
11) 矢田　晋，森田和宏，泓田正雄，平石　亘，青江順一：ダブル配列におけるキャッシュの効率化，第 5 回情報科学技術フォーラム講演論文集（FIT2006），pp.71–72 (2006).
12) untangle. http://www.untangle.com/
13) Clam AntiVirus. http://www.clamav.org/
14) 中村康正，望月久稔：圧縮ディジタル探索木における辞書情報更新の高速化手法，情報処理学会論文誌：データベース，Vol.47, No.SIG13(TOD31), pp.16–27 (2006).

(平成 20 年 3 月 20 日受付)
(平成 20 年 7 月 10 日採録)

（担当編集委員　岩井原 瑞穂）

蔵満　琢麻（学生会員）

昭和 61 年生．平成 20 年大阪教育大学教育学部教養学科情報科学専攻卒業．現在同大学大学院修士課程在学中．情報検索，パターン照合，自然言語処理に関する研究に従事．



松浦　寛生（正会員）

昭和 61 年生．平成 20 年大阪教育大学教育学部教養学科情報科学専攻卒業．同年パナソニック情報システム（株）に入社，現在に至る．情報検索，パターン照合，自然言語処理に関する研究に興味を持つ．

望月　久稔（正会員）

昭和 44 年生．平成 5 年徳島大学工学部知能情報工学科卒業．平成 7 年同大学大学院博士前期課程修了．平成 10 年同大学院博士後期課程修了．博士（工学）．同年大阪府立工業高等専門学校電子情報工学科講師．平成 15 年大阪教育大学教育学部教養学科情報科学講座講師．平成 19 年大阪教育大学教育学部教養学科情報科学講座准教授．現在に至る．情報検索，自然言語処理，知識表現の研究に従事．電子情報通信学会，人工知能学会，自然言語処理学会各会員．