計算機自身にプログラムを作成させる試み*

石田喬也**　安井　裕***　杉山　博***　城　憲三***

まえがき

人工智能実現への努力は，種々の立場からなされてきた．第1の立場は，最も興味深いもので，非常に原始的な情報処理要素の集合を仮定し，それが学習により自己組織化することをねらっているが，その成果は，人工智能実現の目的から見ればまだ微々たるものであるといえる．これに対して，できる限り複雑な情報処理能力を機械に賦与し，人間が問題を解く過程の，Protocol に基づいて人工智能的振舞をさせることを試みる第2の立場がある．この方針に基づく Newell, Simon などの研究は，リスト処理言語の開発により，かなり顕著な成果を報告している[1]．第3の立場は，機械はある程度の情報処理能力を最初から持っていると仮定し，その能力により学習を有効におこない，機械が自らの能力を高めていくことを目的とするものである（たとえば Samuel の Checker Playing Program[1]）．われわれの論文も一つの人工智能の問題を扱っているが，われわれは第3の立場をとっている．すでに報告されている計算機にプログラムを作成させる実験は，第2の立場でなされて学習の余地が残されていないもの[2]や，単に学習の一対象としてとらえたにすぎないもの[3,4,5]である．この論文では，次のようなひとつのシステム，すなわち計算機が外部からの助けを得つつ学習し，徐々にその能力を高め，遂にはある程度未知の問題に対しても，計算機自らがそれを解決するプログラムを正しく作成し得るようになるシステムを作成することを目的としている．したがって，われわれのシステムでは，人工智能そのものとなる計算機（B-計算機と名付ける）とある仮想の計算機(H-計算機と名付ける．B-計算機と一致することもあり得る.）を用意し，与えられた問題を解決するプログラムを H-計算機の言語で B-計算機が作成できるようになることを目指している．ここに H-計算機の言語は任意であり，学習の最初において B-計算機は H-計算機の言語の持つ意味を知らないという，前提をおく．

実際に作成したシステムでは，B-計算機として，NEAC-2206 を選んでいる．このシステム全体のプログラムは約3,000ステップである．このシステムでは H-計算機が1アドレス方式であることを，仮定しているが，特に，H-計算機として**第1表**のような命令を持つ仮想計算機を採用し，加減乗除からなる数式の問題について，実験がおこなわれた．

第1表　実験に使用した H-計算機の命令

命　令	意　味
CAD *n*	(*n*) → *Acc*
CSB *n*	−(*n*) → *Acc*
ADD *n*	(*Acc*)+(*n*) → *Acc*
SUB *n*	(*Acc*)−(*n*) → *Acc*
MUL *n*	(*Acc*)×(*n*) → *Acc*
DIV *n*	(*Acc*)÷(*n*) → *Acc*
STR *n*	(*Acc*) → *n*
（特殊命令）	
LINK *n*	*n*番地から始まるプログラムへのリンク命令
MACR *n*	マクロ命令*n*を実行する
HALT	停止命令

ただし，*Acc*; Accumulator を示す．
(*Acc*), (*n*); それぞれ *Acc*, *n* 番地の内容を示す．

このようなシステムの概略はすでに報告した[6]が，さらに改良を加えたものについて，次節以下に詳細に説明し，最後に実験例を示す．

1. システムの構成

Man（人間）がある Problem（問題）を B-計算機に呈示すると，まず Interpreter が Memory（過去に学習してある事柄）を参照しつつ，Problem の解決，すなわち答となるプログラムの作成を試みる．そこで解決がなされたときは答のプログラムを印字する．解決できなかった場合は，その時点での既知事項を Trainer と Analyzer と Learning Body に報告したのち，Man にその Problem を解くための Advice を要求する．Man より Trainer に Advice が与えられると，Generator が Learning Body の

* An Experiment with a Self-Programming System, by Takaya Ishida, Hiroshi Yasui, Hiroshi Sugiyama and Kenzo Joh (Faculty of Engineering, University of Osaka)
** 大阪大学大学院工学研究科　*** 大阪大学工学部

状態にしたがって，H-計算機のひとつのプログラムを作成し，Trainer がその評価をおこない，Learning Body を教育する．このプログラムの作成，その評価ならびに教育の過程を繰り返えして，遂に Generator が Problem の答となるプログラムを間違いなく作成し得るに至ったとき，Analyzer がいま学習したことを Memory と比較して分析し，その結果を Memory に登録する．以上でひとつの Problem の解決，あるいは学習が終了する．このようにして種々の Problem を学習させることにより，B-計算機は階を追って徐々に能力を高める．

次にこのシステムの各部分を説明する．

1.1 Problem

Problem は次に Backus 記法で示す syntax で記述されなければならない．B-計算機には，予め Problem がこのような syntax でできていることが知らされている．

⟨Problem⟩::=⟨term⟩
⟨term⟩::=⟨unary operator⟩ (⟨operand⟩)|
⟨binary operator⟩ (⟨operand⟩, ⟨operand⟩)
⟨operand⟩::=⟨atom⟩|⟨term⟩
⟨unary operator⟩::=⟨identifier⟩
⟨binary operator⟩::=⟨identifier⟩
⟨atom⟩::=⟨identifier⟩
⟨identifier⟩::=⟨letter⟩|⟨identifier⟩⟨letter⟩|
⟨identifier⟩⟨digit⟩
⟨letter⟩::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|
O|P|Q|R|S|T|U|V|W|X|Y|Z
⟨digit⟩::=0|1|2|3|4|5|6|7|8|9

例

ALGOL の statement

ARITH:=A×(−B1)+C2DE

に対応する Problem は，たとえば次のように記述される．

MOVE(ADD(MULT(A, MINUS(B1)),
C2DE), ARITH)

1.2 Memory

Memory はパターン部と，その定義部が一要素となり，リスト形式で構成されている．パターン部は，atom が operand であることを示す pseudo operand ATOM，あるいは term が operand であることを示す pseudo operand TERM を operand に持つひとつの term である．

定義部はそのパターン部を定義する擬似プログラムからなる．擬似プログラムは term の operand を指示する operand designator，あるいは H-計算機の命令コードと atom の operand を指示する operand designator の一対の順序系列よりできている．あるパターン部を定義する擬似プログラムが2種以上あることを学習した場合，その定義部はそれらの擬似プログラムの OR 結合として登録される（例1）．ここで operand designator を次のように記述する．

d_{0i}; パターン部に表われない atom i を指示する ($i=1, 2, \cdots\cdots$)

d_{x_1}; パターン部中の x_1 番目(左から)の operand を指示する．

$d_{x_1,x_2,\cdots\cdots,x_n}$; $d_{x_1,x_2,\cdots\cdots,x_{n-1}}$ の指示する operand の x_n 番目の operand を指示する．($x_j=1, 2$) ($j=1, 2, \cdots\cdots$)

例 1. パターン部: ADD(ATOM, ATOM)
定義部: {CAD-d_1; ADD-d_2}∪{CAD-d_2; ADD-d_1}

(注) AUB は，AあるいはBいずれであっても良いことを示す．

例 2. パターン部: DIV (TERM, MULT (ATOM, ATOM)) 定義部: {CAD-$d_{2,1}$; MUL-$d_{2,2}$; STR-d_{01}; d_1; DIV-d_{01}}

1.3 Interpreter

まず，Problem 中のすべての identifier の operand を pseudo operand ATOM で置換え，それを subproblem とする．この subproblem と Memory の各要素のパターン部を比較し，一致するものを捜す（ただし，TERM はどのような term とも一致するとする）．

（a） 一致する要素があって，その定義部に term を指示する operand designator がいくつか含まれているとき，その subproblem を定義するプログラムは term を指示する operand designator において決定されずに残る．しかし atom を指示する operand designator については Problem 中の各 identifier との対応づけにより決定される．これらの term を指示する operand designator は subproblem 中においてこれらの指示する term を定義するプログラムのおのおのとして決定される．そこでそれらの term を次に解決すべき subproblem とする．

（b） 一致する要素があって，その定義部に term を指示する operand designator が全く含まれない場合，その subproblem は解決される．すなわち subproblem を定義するプログラムが決定する．

（c） 一致する要素がない場合，その subproblem はこのままでは解決されない．

（a）の場合，新しく作られた subproblem について同様の操作を繰り返えす．かくして subproblem の樹状構造ができ上る．Problem が解決されるためには，それらの subproblem のうち，ある組ではすべての subproblem が同時に解決されることが必要であり（AND 関係），他の組ではいずれかひとつの subproblem が解決されれば良い（OR 関係）．

Problem が解決される場合は，Problem を解決に導いたいくつかの subproblem より Problem が定義され，プログラムを作成することができ，それを印字する．Problem が解決されない場合には，先ずただひとつが解決されれば Problem を解決することができる subproblem を捜し，それに命令 LINK-GPROG（Generator により作成されるプログラムへのリンク命令）を対応させ，Problem 解決に必要な残りのすでに解決済の subproblem にはそれを定義するプログラムを対応させることにより，Problem 解決のための Main Program を作成し，それを印字する．次に Problem を解決するためには解決しなければならない subproblem に対応する Problem の個所を印字する．さらにその subproblem 中の term の operand すべてについて，再び Memory を比較することにより解決を試みる．解決されなければそのままであるが，解決されるものがあるとき，その operand を pseudo operand TERM で置き換え，その operand に定義するプログラムを Learning Body に報告する．これが **1.5** で述べる Interpreter 自らが形づくる state に相当する．このようにして修正されたのちの解決不能の subproblem を Unknown Problem と名付け，それを Analyzer に報告し，かつ印字し，その解決のために Advice を要求する．

1.4 Advice

Advice は Problem を定義づけるに必要十分な情報であれば良いが，同時にその時点での B-計算機が理解できる範囲を越えたものでは意味がない．われわれの実験の初期の学習では，B-計算機は数値情報のみを理解できると考えられる．そこで Trainer は先ず用意してある 10 種類の data location にある数値情報を入れ，次に Problem 中の各 atom に左から順に data location 1, 2, …… を対応させ，Advice を受ける準備をする．そこへ Man は，Trainer に，Advice として，Problem の正しいプログラムが作成され，実行されたと考えたときに得られる結果の，data location の内容を示す goal（GOAL）を与える．さらに，正しいプログラムとアドレス部だけ相違するプログラムが作成され，実行されたと考えたときに得られる結果の data location の内容を示す subgoal（SUBG）を与えることもできる．subgoal を与えるのは単に学習速度を上げるためのものであって，必ずしも必要ではない．このような Advice の方式で，Problem の答となるプログラムを一意的に決定できるためには，最初に各 data location に入れる数値情報を慎重に選ばなければならない．ここではその目的にあう数値情報の基本要素として，素数の平方根を採用する．そこで素数，2, 3, 5, 7, 11, 13, 17, 19, 23, 29 の平方根の近似値が，それぞれ data location 1, 2, ……, 10 に入れられる．特に，Problem 中の atom に対応する data location には，Advice によりこれ以外の数値を入れることもできる（SET）．

例 1.

Problem

MOVE (ADD(AB, CDE), ANS)

に対しては，たとえば次のような Advice を与える．

GOAL (ANS=3.146 AND AB=1.414
AND CDE=1.732)
SUBG (AB=3.146 OR CDE=3.146
OR ANS=3.146)

（注） A AND B は A と B が同時に満足されなければならないことを示す．A OR B は A あるいは B のいずれかが満足されなければならないことを示す．

例 2.

Problem

MOVE (DIV (A, B), C)

に対しては，たとえば次のような Advice を与える．

SET (A=2.449048, B=1.414)
GOAL (C=1.732 AND A=2.449048
AND B=1.414)

1.5 Learning Body

H-計算機の 7 種の命令コード（**CAD, CSB, ADD, SUB, MUL, DIV, STR**），あるいは Interpreter により作成されたマクロ命令，のおのおのをいわゆる推移系における “state” とみなすことにするならば，プログラムの学習の問題は，そのプログラムに対応するこれらの state の推移系列（命令コードの state についてはそのアドレス部も同時）に関する学習の問題と見ることができる．そこで Learning Body は次の 4 種類の行列あるいはベクトルからできている．

（1） 状態推移行列（State transition matrix, STM と略記することにする）

$$\begin{pmatrix} a_{11} & a_{12} & \cdots\cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots\cdots & a_{2n} & b_2 \\ \cdots\cdots & & & & \\ a_{n1} & a_{n2} & \cdots\cdots & a_{nn} & b_n \end{pmatrix} \qquad b_i=\sum_{j=1}^{n} a_{ij} \qquad a_{ij}>0$$

ここで State の総数を n とし state S_i から state S_j への推移確率を $p_{ij}=a_{ij}/b_i$ とする．また，上の推移行列の要素は一般に時間と共に変化するもので，より詳しくは $a_{ij}(t)$ と記すべきものである．

$(i=1, 2, \cdots\cdots, n \quad j=1, 2, \cdots\cdots, n)$

（2） 初期状態決定ベクトル(Initial state finding vector, ISFV と略記することにする)

$(a_{01}, a_{02}, \cdots\cdots, a_{0n}, b_0)$

$$b_0=\sum_{i=1}^{n} a_{0i}, \quad a_{0i}>0$$

initial state-その state よりプログラムの作成を始める state-を state S_i とする確率; $p_{0i}=a_{0i}/b_0$

$(i=1, 2, \cdots\cdots, n)$

（3） 終結状態決定ベクトル（Final state finding vector, FSFV と略記することにする）

$(a_{1f}, a_{2f}, \cdots\cdots, a_{nf}, b_f)$

$$b_f=\sum_{j=1}^{n} a_{jf}, \quad a_{jf}>0$$

final state-その state でプログラムの作成を終了する state-を state S_j とする確率; $p_{jf}=a_{jf}/b_f$

$(j=1, 2, \cdots\cdots, n)$

（4） 命令コード・番地対応行列（Operation code address corresponding matrix, OACM と略記することにする）

$$\begin{pmatrix} c_{11} & c_{12} & \cdots\cdots & c_{1m} & d_1 \\ c_{21} & c_{22} & \cdots\cdots & c_{2m} & d_2 \\ \cdots\cdots & & & & \\ c_{l1} & c_{l2} & \cdots\cdots & c_{lm} & d_l \end{pmatrix} \qquad d_i=\sum_{j=1}^{m} c_{ij} \qquad c_{ij}>0$$

命令コードの state S_i とアドレス AD_j の対応確率; $q_{ij}=c_{ij}/d_i$ $(i=1, 2, \cdots\cdots, l \quad j=1, 2, \cdots\cdots, m)$

ただし，l は命令コードの総数，m は data location の総数を表わす．

このような Learning Body を採用する場合，学習することのできる Unknown Problem は，その解答となるプログラムに対応する state の系列中に，同じ state が2度以上出現しないものに限定される．しかし教育順序に注意し，最も基本的な問題から段階的により高度な問題を学習させていく，という方針をとることにより，この難点を補うことができる．

1.6 Generator

n 次元ベクトル $(p_1, p_2, \cdots\cdots, p_n)$（ここに $0\leqq p_i\leqq 1$, $\sum_{i=1}^{n} p_i=1$）があるとき，乱数 $r(0\leqq r<1)$ を発生し

$$\sum_{i=0}^{t-1} p_i\leqq r<\sum_{i=0}^{t} p_i, \quad p_0=0 \qquad (1)$$

$(t=1, 2, \cdots\cdots, n)$

の関係を満足する t を求めれば，p_t の大きさに比例して t 番目を選択することができる．Generator はこのような選択法則にしたがって，Learning Body の各値よりひとつの H-計算機のプログラムを作成することができる．

まず，ISFV, FSFV の各要素の値に応じて，それぞれ initial state, final state を決め，次に STM の各要素の値に応じて initial state から final state まで state の推移系列を実現させる．その間に命令コードの state を選択した場合，その state に対応するアドレスを OACM より決定する操作を加え，final state に到達したときプログラムの作成を終了する．このようにして Generator により作成されたプログラムを Generated Program と名付る．

1.7 Trainer

Trainer はまず Main Program を実行する．Main Program 中には Generated Program へのリンク命令が必ずひとつ含まれているので，結局は，Generated Program が実行されることにもなる．次に Main Program が実行された後の各 data location の内容を，すでに与えられている Advice と比較し，goal が達成されているか（success），subgoal が達成されているか（subsuccess），いずれも達成されていないか（failure）を判定する．

この結果 success の場合には，Generated Program のたどった state の推移，ならびに命令コードの state とアドレスの対応，のすべてについてその推移確率，あるいは対応確率を上げる操作を加える．

subsuccess の場合，Generated Program のたどった state の推移すべてについて，その推移確率を上げ，命令コードの state とアドレスの対応すべてについては，その対応確率を下げる．

また failure の場合には，Generated Program のたどった state の推移すべてについて，その推移確率を下げる．

ISFV, FSFV には，success あるいは subsuccess の場合に特別の教育をする．

Learning Body が学習する過程の詳細は 2. で説

明する．

Trainer は通常このような Learning Body の教育をおこなって後，再び Generator に仕事を移すが，特に success であって，その Generated Program のたどったすべての state の系列についての推移確率と，initial state 並びに final state が選ばれた確率との積，および，すべての命令コードの state とアドレスの対応の対応確率の積，が共に十分1に近い場合には，Unknown Problem の学習が一応完了したとみなし，Analyzer に仕事を移す．

1.8 Analyzer

Analyzer は Unknown Problem UP を，Memory 中のひとつの要素 M_i のパターン部と比較し，対応するひとつの term が相違すれば（両者の term の operator が一致しないか，両者の operator は一致するが，第1 operand も第2 operand も共に一致しない場合），UP，M_i のその term をそれぞれ抽出する．もしそれらが UP そのもの，あるいは M_i のパターン部そのものであれば，UP と他の要素との比較に移る．そうでなければ，UP と M_i の相違する term はある共通の operator の第 j operand ($j=1$ あるいは2）になっているから，UP と M_i のパターン部の共通部分は，相違する operand を，pseudo operand TERM で置換えたものである．かくして，UP と M_i のパターン部の共通部分と相違部を抽出できるから，次に，これらのパターン部の定義部を求める．そのために Generator と Trainer により学習した UP の答となるプログラムと M_i の定義部であるプログラムを UP と M_i のパターン部の共通部分の対応関係を考慮して比較し，両プログラムで共通に存在する部分とそれ以外の部分を抽出する．以上の抽出操作の後，Analyzer は次の3要素を Memory に登録する．

1. パターン部；UP と M_i のパターン部の共通部分
   定義部；UP と M_i のプログラムの共通部分
2. パターン部；UP のパターン部中の，M_i のパターン部と相違している部分
   定義部；UP のプログラム中の，M_i のプログラムと相違している部分
3. パターン部；M_i のパターン部の中で，UP のパターン部と相違している部分
   定義部；M_i のプログラムの中で，UP のプログラムと相違している部分

Memory 中のどの要素とも共通部を見出せなかった Unknown Problem に対しては，それをパターン部とし，学習したプログラムを定義部とする1要素を作り，Memory に登録する．

2. 学習過程

ここでは Learning Body 各部の学習について述べる．しかし，STM と OACM, ISFV と FSFV はそれぞれ類似の学習方式をとっているので，OACM と FSFV の学習の説明は紙面の都合上省略する．以下の記述では，Generated Program の長さを L, そのたどった state の推移系列を，$S_{t_1} \to S_{t_2} \to \cdots\cdots \to S_{t_L}$ と表わすことにする．

2.1 State transition matrix の学習

Generated program が success, あるいは subsuccess であるとき，次のような学習をおこなう．

$$\begin{matrix} a_{t_i t_{i+1}} \longrightarrow a_{t_i t_{i+1}} + b_{t_i} \cdot k_S \\ b_{t_i} \longrightarrow b_{t_i} + b_{t_i} \cdot k_S \end{matrix} \quad (i=1, 2, \cdots\cdots, L-1) \qquad (2)$$

ただし，$k_S > 0$

この結果を推移確率の変更として見ると

$1/(1+k_S)=\alpha$ とおくことにより　　(3)

$$\begin{cases} p_{t_i t_{i+1}} \longrightarrow \alpha p_{t_i t_{i+1}} + (1-\alpha) \\ p_{t_i j} \longrightarrow \alpha p_{t_i j} \quad (j=1, 2, \cdots\cdots, n \neq t_{i+1}) \end{cases} \qquad (4)$$

$(i=1, 2, \cdots\cdots, L-1)$

となる．これは state 間の推移確率に linear operator が作用されたことを示す．

一般的にいって，確率を変化させるために作用する operator として種々考えられるが，学習の性質をよく近似し，かつ解析が比較的容易であるという点から前述のような linear operator を採用した．直接（4）のような形をとらず（2）のようにしたのは，両者の演算回数の違いより知られるように linear operator の作用時間をできるだけ短くしたい目的からである．確率を成分とする n 次元縦ベクトル P(成分；p_i, $\sum_{i=1}^{n} p_i = 1$) の各成分を変化させるための linear operator として MOSTELLER 流の統計的学習理論にしたがい，つぎの行列 T を用いることにする．

$$\mathbf{T} = a\mathbf{I} + (1-a)\mathbf{\Lambda} \qquad (5)$$

ただし，$0 \leqq a \leqq 1$

$\mathbf{I}$；(n, n) 単位行列

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \lambda_1 \cdots\cdots \lambda_1 \\ \lambda_2 & \lambda_2 \cdots\cdots \lambda_2 \\ \cdots\cdots\cdots\cdots\cdots \\ \lambda_n & \lambda_n \cdots\cdots \lambda_n \end{pmatrix} \qquad \begin{matrix} 0 \leqq \lambda_k \leqq 1 \\ \sum_{k=1}^{n} \lambda_k = 1 \end{matrix}$$

このTをPに作用すると，

$$\mathbf{TP}=a\mathbf{P}+(1-a)\boldsymbol{\lambda} \qquad (6)$$

$$\boldsymbol{\lambda}=(\lambda_1, \lambda_2, \cdots\cdots, \lambda_n)$$

であるから，この結果各成分は

$$p_i \rightarrow ap_i+(1-a)\lambda_i \qquad (7)$$

$$(i=1, 2, \cdots\cdots, n)$$

という変化をする．またTをPにN回繰り返し作用すれば，

$$\mathbf{T}^N\mathbf{P}=a^N\mathbf{P}+(1-a^N)\boldsymbol{\lambda} \qquad (8)$$

であるから，$0<a<1$ の場合，Nが大きくなるにつれ $\mathbf{T}^N\mathbf{P}$ は $\boldsymbol{\lambda}$ に漸近する．すなわち $\mathbf{T}^N\mathbf{P}$ の i 成分は λ_i に漸近する．

（7）と（4）を比較することにより，（4）は，Generated Programのたどったすべての系列の推移確率に，その値を1に漸近させるoperatorを作用していることを知る．

また，Generated Programがfailureであるときは，つぎのような学習をおこなう．

$$\begin{cases} a_{t_it_{i+1}} \rightarrow a_{t_it_{i+1}}-b_{t_i}\cdot k_f \\ b_{t_i} \rightarrow b_{t_i}-b_{t_i}\cdot k_f \end{cases} \qquad (9)$$

特に $a_{t_it_{i+1}}-b_{t_i}k_f \leqq 0$ のときは

$$a_{t_it_{i+1}} \rightarrow a_{t_it_{i+1}},\ b_{t_i} \rightarrow b_{t_i} \qquad (10)$$

ただし，$0<k_f<1$ $(i=1, 2, \cdots\cdots, L-1)$

（9）を推移確率の変化として表わすと

$$\begin{cases} p_{t_it_{i+1}} \rightarrow \beta p_{t_it_{i+1}}+(1-\beta) \\ p_{t_ij} \rightarrow \beta p_{t_ij} \quad (j=1, 2, \cdots\cdots, n \neq t_{i+1}) \end{cases} \qquad (11)$$

$$\beta=\frac{1}{1-k_f} \qquad (12)$$

$$(i=1, 2, \cdots\cdots, L-1)$$

であるから，やはりlinear operatorが作用されることになる．しかし(11)を実現するlinear operatorは，(12)から $\beta>1$ のため（4）の場合と異り（5）の a の条件を満足しないので，確率を，変化させるlinear operatorとしては不十分である．そこで(10)のような学習を用意しているのである．

Generated Programがsuccess (subsuccess) であるとは，それに対応するstateの推移系列中に，state S_j から無駄なstateへの推移も含め，state S_j からある順序に推移して後state S_k に至る（その確率を $\overline{p_{jk}}$ とする）推移系列 $\overline{S_{jk}}$ を含むことであるとすれば，このような学習方式をとることにより，十分多数回の試行（Generatorがひとつのプログラムを作成することを1回の試行とする）の後には $\overline{p_{jk}} \rightarrow 1$ となることが次のようにして証明される．ここで無駄なstateとは，stateの推移系列中にその stateを含んでいても含んでいなくても，その対応するプログラムを実行した結果においては変わりがない stateのことをいう．

（証明）failureのときの学習は，k_f に後述のように非常に小さい値が採用されているので，success (subsuccess) の場合の学習に比し無視することができる．そこで以下の証明では success (subsuccess) のときのみ（4）の形の学習をおこなうものとする．

さて，$\overline{S_{jk}}$ は無駄なstateを全く含まない場合でも，任意のProblemに対して一意的に決まるとは限らない．そこで一般に $\overline{S_{jk}}$ の中で，ひとつのstate S_a から他の r 個のstate S_{b_i} $(i=1, 2, \cdots\cdots, r)$ への推移が許される場合を考える．$r=1$ の場合は，success (subsuccess) のとき必ず $S_a \rightarrow S_{b_1}$ の推移がなされることになり，(8) よりsuccess(subsuccess)の回数が増すにつれ $\overline{p_{ab_1}}$ は必ず1に漸近する．$r \geqq 2$ の場合は，十分多数回のsuccess (subsuccess) の後を考えると

$$\sum_{i=1}^{r} p_{ab_i} \simeq 1 \qquad (13)$$

となっているから，1回のsuccess (subsuccess) につき $S_a \rightarrow S_{b_i}$ の推移確率は，確率 p_{ab_i} でつぎのように operator Q_i の作用をうけて

$$Q_i p_{ab_i}=\alpha_i p_{ab_i}+1-\alpha_i \qquad (14)$$

となり，確率 $(1-p_{ab_i})$ で，operator $\bar{Q}_i$ の作用をうけて

$$\bar{Q}_i p_{ab_i}=\bar{\alpha}_i p_{ab_i} \qquad (15)$$

となる．したがって (13) のようになって，後から数えて n 回目のsuccess (subsuccess) の後の，$S_a \rightarrow S_{b_i}$ の推移確率の k 次のモーメントを $\mathrm{V}_{k,n}(p_{ab_i})$ とすれば，次の漸化式が成り立つ．〔7）第4章参照〕

$$\mathrm{V}_{1,n}(p_{ab_i})=(1-\alpha_i+\bar{\alpha}_i)\mathrm{V}_{1,n-1}(p_{ab_i}) +(\alpha_i-\bar{\alpha}_i)\mathrm{V}_{2,n-1}(p_{ab_i}) \qquad (16)$$

$$\mathrm{V}_{2,n}(p_{ab_i})=(1-\alpha_i)^2\mathrm{V}_{1,n-1}(p_{ab_i}) +\{2\alpha_i(1-\alpha_i)+\bar{\alpha}_i^2\}\mathrm{V}_{2,n-1}(p_{ab_i}) +(\alpha_i^2-\bar{\alpha}_i^2)\mathrm{V}_{3,n-1}(p_{ab_i}) \qquad (17)$$

(16)，(17) で $n \rightarrow \infty$ とすれば

$$\mathrm{V}_{1,\infty}(p_{ab_i})=\mathrm{V}_{2,\infty}(p_{ab_i}) \qquad (18)$$

の関係を得る．純粋不連続な分布では1次のモーメントと2次のモーメントが一致する分布は，点1，0においてのみ密度を持つ2項分布であり，点1における密度は1次のモーメントである．〔7）pp. 155～156 参照〕

以上のことより，$S_a \rightarrow S_{b_i}$ の推移確率は，十分多

数回の success (subsuccess) の後には $V_{1,\infty}(p_{ab_i})$ の確率で1に漸近することがわかる．一方，このとき (13) の関係が満たされているから，state S_a から起こりうる r 個の推移のうち，そのいずれか一つの推移確率が必ず1に漸近することになる．この考慮で，$S_a \to S_{b_i}$ は $\overline{S_{jk}}$ 中の任意のひとつの推移で良かったから，十分多数回の success (subsuccess) を含む十分多数回の試行の後には，$\overline{S_{jk}}$ としてとり得るあらゆる state の推移系列のうちのひとつを必ず学習するに至る（$\overline{p_{jk}} \to 1$）ことが証明されたことになる．

（証明終り）

パラメータ k_S については次のように考察した．ある Unknown Problem に対する success (subsuccess) の Generated Program の最小の長さを L_m とすれば，$L_m \geqq 2$ のとき，$L \geqq L_m$ の success (subsuccess) の Generated Program のたどった state の推移のうち，約 $(L_m-1)/(L-1)$ のものが success (subsuccess) に寄与している．そこで（4）より

$$1-\alpha = k_S' \cdot \frac{L_m-1}{L-1} \qquad (19)$$

$$0 < k_S' \leqq 1$$

とすることが考えられる．しかし L_m は未知であるから，最小の値2で代用する．$L=1$ したがって $L_m=1$ のときは明らかに STM が関与する必要はない．(19) と（3）より

$$k_S = \frac{k_S'}{L-1-k_S'}, \quad \alpha = 1-\frac{k_S'}{L-1} \qquad (20)$$

となる．k_S' は学習の早さの点からでは，大きい程良く，また無駄な state への推移を含むプログラムを学習してしまうことを防ぐためには小さい程良い．このことは実験によって確かめられた．Unknown Problem が MINUS (TERM) の場合（$L_m=3$, state の総数=8）の実際の実験結果を第2表に示す．パラメータ k_f は学習すべきプログラムの長さが短い場合には，その値が大きいことが学習速度向上に役立つが，プログラムが長い場合では，学習速度のばらつきを大きくする原因になるので，一般的には十分小さい値を採用することが望ましい．

2.2 Initial finding vector の学習

過去に作成された success (subsuccess) の Generated Program のうちの最小の長さのものと，等しいかあるいは，それより短い長さの success (subsuccess) の Generated Program が報告されたときのみについて，次のような学習をおこなう．

$$\begin{cases} a_{0t_1} \to 1, \ a_{0t_i} \to \prod_{j=1}^{i-1} (1-p_{t_j t_{j+1}}) \\ b_0 \to 1+\sum_{i=2}^{L} \prod_{j=1}^{i-1} (1-p_{t_j t_{j+1}}) \end{cases} \qquad (21)$$

この結果，initial state を S_{t_i} とする確率は

$$p_{0t_i} = p_{0t_{i-1}}(1-p_{t_{i-1}t_i}) = p_{0t_1} \prod_{j=1}^{i-1}(1-p_{t_j t_{j+1}})$$

$$p_{0t_1} = 1/1+\sum_{i=2}^{L} \prod_{j=1}^{i-1}(1-p_{t_j t_{j+1}}) \qquad (22)$$

となる．

ひとつの success (subsuccess) のプログラムに対応する state の推移系列のうち，結果に有効なのは，$\overline{S_{jk}}$ というその中の一部の推移系列であるから，success (subsuccess) のたびに，$S_{t_i}(i=1, 2, \cdots\cdots, L)$ のうちいずれが真の S_j に相当するかを推定することが必要である．ここでは，S_{t_i} を S_j と推定する確率と，$S_{t_{i-1}}$ を S_j と推定する確率の比は $(1-p_{t_{i-1}t_i})$ であると仮定している．このような仮定をとることにより，$S_{t_{i-1}} \to S_{t_i}$ の推移確率が大きくなる程，S_{t_i} を S_j と推定する確率が減少し，したがって $S_{t_{i-1}}$ を S_j と推定する確率が増加することになる．

このような学習方式をとることにより，ISFV の学習が収束することは次のようにして証明される．

（証明）　前述のように多数回の試行の後には $\overline{p_{jk}} \to 1$ となるから，$S_{t_i}(t_i \leqq j)$ を S_j と推定する確率以外は0に漸近する．すなわち，この状態では initial state として S_j，あるいは推移系列 $\overline{S_{jk}}$ 中に含まれない state のみが選ばれ得る．initial state として S_j が選ばれた場合は，ただちに $p_{0j} \fallingdotseq 1$ となるから ISFV の学習は完了する．S_j

第2表　MINUS (TERM) の実験結果

k_S'	α	$N=\frac{1}{20}\sum_{i=1}^{20} N_i$	$\sigma=\sqrt{\frac{1}{20}\sum_{i=1}^{20}(N_i-\bar{N})^2}$	$\bar{T}=\frac{1}{20}\sum_{i=1}^{20} T_i$	$\bar{E}=\frac{1}{20}\sum_{i=1}^{20} E_i$
0.8	0.8	965	189	2.2分	9/20
0.4	0.9	1037	188	2.4	5/20
0.2	0.95	1192	180	2.7	0/20
0.1	0.975	1401	155	2.9	0/20

ただし，

N_i; 第 i 回目の実験において学習完了までにプログラムを作成した回数（試行回数）

T_i; 第 i 回目の実験において学習完了までに要した時間

E_i; 第 i 回目の実験において学習したプログラムに含まれている無駄な state の数 ($i=1, 2, \cdots\cdots, 20$)

以外の state が initial state に選ばれても，通常 p_{0j} の値は正に保たれるが，ある Generated Program により，$p_{0j}\fallingdotseq 0$ となることがあるとすれば，(22)より $p_{t_{i-1}t_i}\fallingdotseq 1$ である推移，$S_{t_{i-1}}\to S_{t_i}(t_i\leqq j)$ が少なくともひとつ存在することになる．そのときは $S_{t_{i-1}}$ と S_{t_i} は等価となるから，initial state として選ばれ得る state の数は実質上，それ以前よりもすくなくともひとつ減少する（p_{0j} は再び正の値に回復し得る）．これを繰り返えすと initial state に選ばれ得る state の数は単調減少するが，state の数は有限であるから必ず initial state がひとつ決定されるに至ることを知る．（証明終り）

2.3 学習過程のまとめ

Learning Body のある状態において，Generator が success (subsuccess) のプログラムを作成する確率，p_{suc} は一般に次式で与えられる．

$$p_{suc}=(p_{0j}+\sum_{i\neq k}p_{0i}h_{ij}^{(k)})\bar{p}_{jk}\cdot p_{kf}+\sum_{S_l\in S_{jk}}(p_{0j}+\sum_{i\neq l}p_{0i}\cdot h_{ij}^{(l)})\cdot\bar{p}_{jk}\cdot b_{kl}\cdot p_{lf}\quad(23)$$

ただし，(以下の表現については 8) 参照)

$h_{ij}^{(k)}$; state S_i から state S_k を通ることなく，state S_j に一度以上推移する確率

b_{ij}; state S_i から初めて state S_j に推移する確率

$\mathbf{H}_k=(h_{ij}^{(k)})=(\mathbf{N}_k-\mathbf{I})\mathbf{N}_{kd_0}^{-1}$

$\mathbf{B}_i=(b_{ij})=\mathbf{N}_j\cdot\mathbf{R}_j\quad\mathbf{N}_k=(\mathbf{I}-\mathbf{Q}_k)^{-1}$

$\mathbf{Q}_k$; 推移確率行列 $\mathbf{P}=(p_{ij})$ より k 行，k 列の成分を除いた $(n-1, n-1)$ 行列．n; state の総数

$\mathbf{N}_{kd_0}$; $\mathbf{N}_k$ の対角成分のみよりなる $(n-1, n-1)$ 対角行列．

$\mathbf{R}_j$; $\mathbf{P}$ の j 列より j 成分を除いた $(n-1)$ 次元縦ベクトル

$\mathbf{I}$; $(n-1, n-1)$ 単位行列

すでに証明した $\overline{p_{jk}}$ ならびに ISFV, FSFV の学習の収束性より，十分多数回の試行の後には，$p_{suc}\to 1$ となることは明らかであろう．さらに OACM の収束性も考慮に入れれば，以上述べた学習過程のまとめとして次のことが言える．

このような学習過程をとることにより，十分多数回の試行の後には，必ずひとつの Unknown Problem の答となるプログラムを学習するに至る．常にそれが Unknown Problem の答となるプログラムのうちの最小の長さのものであるとは限らないが，学習過程のパラメータを適当に選ぶことにより，学習速度の犠牲において，その確率を，十分1に近づけることができる．

3. 実 験 例

binary operator, MOVE, ADD, SUB と，unary operator, MINUS, PLUS を学習させる実験の一例では，操作時間も含めて約65分で，これらの operator すべてを学習させることができた．その後 B-計算機はこれら五つの operator に関しては，自立状態にあり，たとえば次のような Problem (P 1)

MOVE (SUB(ADD(SUB(MINUS(ADD(ABCD, PLUS(EFG))), SUB(MINUS(H 1), H 2)), IJ), ADD(KLM, N)), ANSWER)

を呈示すると，**第1図**のような (P 1) のプログラムを，約2.5秒（読み込み時間も含める）で，印字する（解決する）．ところが未知の operator DIV を含む

```
MPROG
CAD     N
ADD     KLM
STR     WORK 1
CSB     H 1
SUB     H 2
STR     WORK 2
CAD     EFG
ADD     ABCD
STR     WORK 3
CSB     WORK 3
SUB     WORK 2
ADD     IJ
SUB     WORK 1
STR     ANSWER
HALT
```

(注) WORK i (i=1, 2, 3) は作業番地を示す

第1図 Problem (P 1) を呈示したときの印字

Problem (P 2)

MOVE(SUB(ADD(SUB(MINUS(DIV(ABCD, EFG)), SUB(MINUS(H 1), (H 2)), IJ), ADD (KLM, N)), ANSWER)

を呈示すると，**第2図**のような印字をし，DIV に対する Advice を要求して一旦止る．つぎに未知の operator DIV を学習するための Problem (P 3) と Advice として

MOVE (DIV(A, B), C)

SET (A # 512449048, B # 511414)

```
      MPROG
CAD     N
ADD     KLM
STR     WORK 1
CSB     H 1
SUB     H 2
STR     WORK 2
LINK    GPROG
STR     WORK 3
CSB     WORK 3
SUB     WORK 2
ADD     I J
SUB     WORK 1
STR     ANSWER
HALT
    UNKNOWN
DIV (ABCD, EFG)
DIV(ATOM, ATOM)
    ADVICE
```

未知の operator DIV を含む Problem (P 2) を与えて得られた計算機の応答.

```
MOVE (DIV (A, B), C)
```

未知の operator DIV (ATOM, ATOM) を学習するための Problem (P 3) を与える.

```
      MPROG
LINK    GPROG
STR     C
HALT
    UNKNOWN
DIV(A, B)
DIV(ATOM, ATOM)
    ADVICE
```

(P 3) に対する応答.

```
SET (A # 512449048, B # 511414)
GOAL (C # 511732. A # 512449048.
B # 511414)
¥
```

(P 3) のための Advice を与える.

```
    LEARNED
```

約 24 秒で学習を完了する.

```
MOVE(SUB(ADD(SUB(MINUS(DIV(ABCD, EFG)), SUB
(MINUS(H 1), H 2)), IJ), ADD(KLM, N)), ANSWER)
```

学習後 (P 2) を計算機に呈示する.

```
      MPROG
CAD     N
ADD     KLM
STR     WORK 1
CSB     H 1
SUB     H 2
STR     WORK 2
CAD     ABCD
DIV     EFG
STR     WORK 3
CSB     WORK 3
SUB     WORK 2
ADD     IJ
SUB     WORK 1
STR     ANSWER
HALT
```

約 2.5 秒で (P 2) の解となるプログラムを印刷する.

第 2 図　Problem (P 2) に含まれる未知の operator を学習させて (P 2) のプログラムを得るまでの過程とその印字

```
GOAL (C # 511732. A # 512449048.
B # 511414)
¥
```

を第 2 図のように与えると約 24 秒で学習を完了する. この状態において改めて Problem (P 2) を呈示すると約 2.5 秒で解となるプログラムが得られる.

むすび

ここに報告した, 実際にわれわれの作成したシステムは, コンパイラーを作り出すシステムのひとつであるとみなすこともできる. このシステムにより, B-計算機は Problem の syntax に対応する H-計算機の言語で表現される semantics を学習することになり, 一通りの事柄を学習した後には, 一種の syntax directed compiler[9] になっている.

この論文での実験をより一般的な形に拡張するためには, まず学習過程に対して, 1.5 に触れたように, time dependent な学習が可能になるように考慮が払われなければならない. なぜなら一般の学習では, ある state S_i から, 推移する先の state は, time (その state S_i が推移系列中のいくつかの state S_i のうちの何番目に表われたものか) を考慮して始めて一意的に決定され得るからである. その他, Problem の拡張, それに伴なう Advice の高度化, また Analyzer の改良など, システムを一般的なものに拡張するにはまだまだ多くの問題が残されているようである. さらに与えられた任意の問題を解決する人工智能を実現したい目的からは, H-計算機が持つべき言語の検討が必要である.

参考文献

1) Feigenbaum, E.A. and Feldman, J. (eds): Computers and thought. McGraw-Hill, 1963.
2) Simon, H.A.: Experiments with a Heuristic Compiler. J. ACM **10**, 4 (Oct. 1963).
3) Friedberg, R.M.: A learning machine, Part I. IBM J. Res.

Dev. **2**, 1 (Jan. 1958).
4) Friedberg, R.M., Dunham, B. and North, J.H.: A learning machine, part II. IBM J. Res. Dev. **3**, 3 (July 1959).
5) Kilburn, T., Grinsdale R.L. and Sumner, H.A.: Experiments in machine learning and thinking. In Proc. Internat. Conf. on Inf. Proc., UNESCO Paris. 1959.
6) 石田，安井，杉山，城：計算機自身にプログラムを作成させる試み，昭和40年度情報処理学会大会講演予稿集.
7) Bush, R.R. and Mosteller, F.: Stochastic Models For Learning. John Wiley & Sons, Inc., New York, N.Y. 1955.
8) Kemeny, J.G. and Snell, J.L.: Finite Markov Chains. D. Van Nostrand Company, Inc., Princeton, New Jersey, 1960.
9) Irons, E.T.: The Structure and Use of the Syntax Directed Compiler. In Third Annual Review of Automatic Programming, Pergamon, New York, 1963.

（昭和41年7月18日受付）