A Self-Programing System with Experiments

TAKAYA ISHIDA*, HIROSHI YASUI*, HIROSHI SUGIYAMA*
AND KENZO JOH*

Introduction

It may be really challanging to imagine a computer which analyzes problems and generates programs by itself to solve them.

In this paper, we describe a certain self-programming system by which a computer gradually becomes clever using his past experiences, by the statistical learning procedure of Bush-Mosteller type[1], with aids of Man and finally he grows up to be able to solve some problems which he has never learned yet.

Approaches which have been tried toward our fascinating goal, ranges from the heuristic one[2] to the deterministic one[3] (i. e. automatic selection of suitable procedures).

Even in the heuristic approach, the computer is supposed to have been given highly organized power in advance. Incidentally, it must be more fascinating to imagine a general system i. e. self-programming system which educates a computer to acquire such power and bring up the computer to a programmer of any object computer.

In order to realize such a purpose, we prepared two computers: one is a programmer in embryo (called B-computer) and the other is a hypothetical computer (called H-computer) on which B-computer generates a program to solve a problem.

Table 1. The language of the H-computer in our Experiments.

Instruction	Function
CAD n	$(n) \rightarrow Acc$
CSB n	$-(n) \rightarrow Acc$
ADD n	$(Acc)+(n) \rightarrow Acc$
SUB n	$(Acc)-(n) \rightarrow Acc$
MUL n	$(Acc)\times(n) \rightarrow Acc$
DIV n	$(Acc)\div(n) \rightarrow Acc$
STR n	$(Acc) \rightarrow n$
(*Special*)	
LINK n	links to the program starting from address n
MACR n	excute the macro instruction n
HALT	halts the program

Note. Acc: Accumulator
(Acc), (n): the contents of Acc and address n respectively

This paper first appeared in Japanese in Joho Shori (the Journal of the Information Processing Society of Japan), Vol. 8, No. 3 (1967), pp. 121–130.
* The authors of the above paper were T. Ishida, H. Yasui, H. Sugiyama and K. Joh, Faculty of Engineering, Osaka University. Presently, T. Ishida is at the Kamakura Works, Mitsubishi Electric Corp. K. Joh is professor of mathematics at the Faculty of Engineering, Kansai University.

Both computers may be chosen arbitrarily. As a special system, suppose they are the same, then the B-computer will finally aquire an artificial intelligence which enables to answer the solution of a given problem.

The basic assumption underlying our self-programming system is that our B-computer does not know the meaning of the language of the H-computer at the beginning.

Then, in order to make our experiments easier, we restricted our problems to those which can be handled by the B-computer. The problems are composed of the arithmetic operations such as addition, subtraction, multiplication and division. The language of the H-computer is shown in Table 1.

1. *Structure of the Self-programming System*

When *Man* shows a *Problem* to the B-computer, *Interpreter* tries to solve it, refering to *Memory*, that is, to create a program, written by the language of the H-computer, the execution of which may offer the answer of the *Problem*.

Then, if the *Problem* is solved, *Interpreter* shows the *Answer*. If it cannot be solved, *Interpreter* informs *Trainer*, *Analyzer*, and the *Learning Body* of the knowledge about the *Problem* and requests *Man* to supply *Advice* for solving the *Problem*.

After *Trainer* received *Advice* from *Man*, *Generator* generates one program in accordance with the status of *Learning Body*. Then the *Trainer* evaluates this program on the basis of the *Advice* and trains the *Learning Body*.

These processes of program generation, evaluation, and training are iterated until the *Generator* construct successfully a correct program, then the *Analyzer* compares the matters just learned with the contents of *Memory* and then registers the comparison into the *Memory*.

Accumulating such learning on various *Problems*, our B-computer becomes clever. In the following sections, the components of the system will be described in detail.

1. 1. *Problem*

Problem should be expressed in the following syntax

⟨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

1. 2. *Memory*

One element of *Memory* is composed of the pattern field and the definition field. The syntax of the pattern field is the same with that of *Problem* with the replacement of $\langle$operand$\rangle$ ::= $\langle$atom$\rangle$ | $\langle$term$\rangle$ by $\langle$operand$\rangle$ ::= ATOM | TERM
where ATOM and TERM are pseudo operands. The definition field consists of a sequence of operand designators for TERM pseudo operands or pairs of operation codes and operand designators for ATOM pseudo oprands.

1. 3. *Interpreter*

Interpreter first creates a subproblem by replacing all $\langle$atom$\rangle$ operands in *Problem* with ATOM pseudo operands. Comparing this subproblem with each element in *Memory*, it tries to find an element which is equal or matches with the subproblem in the pattern field ($\langle$term$\rangle$ operand matches with a TERM pseudo operand).

(1) When there is a matching element (and there is no equal element), *Interpreter* sets up another subproblems next to be solved by extracting $\langle$term$\rangle$ operands corresponding to the TERM pseudo operands from the subproblem.

(2) When there is an equal element, the solution of the subproblem can be obtained from the definition field of the element.

(3) If there is no equal or matching element, the effort of solving the subproblem is continued no more.

Thus, the *Interpreter* forms a tree of subproblems connected by AND or OR relation until the time he obtains the solution.

If he fails to solve the *Problem*, he informs the *Man*, of the *Main Program* of the *Problem* and also of the subproblem which makes the *Problem* unsolvable. When the subproblem itself consists of a tree of subproblems, the $\langle$term$\rangle$ oprands in the subproblem corresponding to solvably composed subproblems are replaced by TERM pseudo operands. The subproblem after this modification is called an *Unknown Problem*. The solution programs of the composing subproblems are sent to *Learning Body* as the macro states generated by *Interpreter*. Finally *Interpreter* requests *Advice*.

1. 4. *Advice*

Advice should bo composed of necessary and sufficient information to define the *Problem*. Naturally, it is meaningless if the B-computer at the time cannot understand this information. At the infant learning stage, we may supose that the B-computer is able to understand only numerical information. Hence, *Trainer* first insert certain numerical information into some specially prepared locations which correspond to $\langle$atom$\rangle$ operands in the *Problem* one by one and wait for the *Advice*.

There are two kinds of *Advice*: *Goal* and *Subgoal*. The latter is given to speed up learning and not always necessary. *Goal* is the the information which shows the status (the contents of those location) after the execution of solution program of the *Problem*. *Subgoal* is composed of partial information of *Goal*.

1. 5. *Learning Body*

In the sequel, taking each operation code of the H-computer as a "state" (operation code state) and using the macro states generated by *Interpreter*, we are able to regard the learning process of a program as that of a transition sequence of states which corresponds to the program. Hence, *Learning Body* consists of two matrices and two vectors as follows.

(1) State transition matrix (STM)

$$\begin{pmatrix} a_{11} & a_{12}\cdots\cdots a_{1n} & b_1 \\ a_{21} & a_{22}\cdots\cdots a_{2n} & b_2 \\ \cdots & \cdots & \cdots \\ a_{n1} & a_{n2}\cdots\cdots a_{nn} & b_n \end{pmatrix} \qquad d_i=\sum_{j=1}^{m} a_{ij} \qquad a_{ij}>0 \qquad (i=1, 2, \cdots\cdots n \quad j=1, 2, \cdots\cdots, n).$$

Where n is the number of states. The transition probability with which the process moves from state S_i to state S_j is expressed by

$$p_{ij}=a_{ij}/b_i.$$

(2) Initial state finding vector (ISFV)

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

$$b_0=\sum_{i=1}^{n} a_{0i}, \quad a_{0i}>0 \qquad (i=1, 2, \cdots\cdots, n)$$

where, the probalility that the initial state (from which the process starts) is S_i is

$$p_{0i}=a_{0i}/b_0.$$

(3) Final state finding vector (FSFV)

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

$$b_f=\sum_{j=1}^{n} a_{jf}, \quad a_{jf}>0 \qquad (j=1, 2, \cdots\cdots, n)$$

where the probalility that the final state (at which the process ends) is S_j is

$$p_{jf}=a_{jf}/b_f.$$

(4) Operation code address corresponding matrix (OACM)

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

where l is the total number of operation codes, and m is the total number of data locations, and the probability that the address AD_j is corresponding to the operation code state S_i is expressed as

$$g_{ij}=c_{ij}/d_i.$$

1. 6. *Generator*

Suppose there are n branching ways from a certain branch point and the probability to select i-th way is p_t, then we can select the t-th way in proportion to P_t, by generating a random number $r(0\leq r<1)$ and determinant satisfying the following relationship,

$$\sum_{i=0}^{t-1} p_i \leq r \leq \sum_{i=0}^{t} p_i, \quad p_0=0 \tag{1}$$

where $t=1, 2, \cdots\cdots, n.$

Based upon such a procedure, *Generator* generates a program of H-computer (*Generated Program*) as follows. Firstly, we determine an initial state and a final state from ISFV and FSFV, and then we obtain a sequence of state trasitions by STM, starting from the initial state, ending at the final state. Then, each time an operation code state is selected, the address corresponding to the state is determined by OACM.

1. 7. *Trainer*

Trainer executes the *Main Program* for the first time. Since is the *Main Program* there is contained at least one link instruction tc the *Generated Program*, the *Generated Program* is excecuted. Then, *Trainer* decides one out of the following threc cases, that is, the *Goal* has been attained (called *Success*), some *Subgoals* has been attained (called *Subsuccess*), and none of them has bee attained (called *Failure*). For all transitions in the *Generated Program* the transition probabilities are increased in case of the *Success* or the *Subsuccess* and decreased in case of the *Failure*. The details of reinforcing procedure will be mentioned in Section 2. *Trainer* usually switches his work to the *Generator* after finishing the education of the *Learning Body*, except the occasion where all the following three conditions are satisfied simultaneously. When the condition are met, the *Trainer* judges the learning has been completed and switches his work to the *Analyzer*.

(1) The *Generated Program* has been judged as successful.

(2) The product of transition probabilities of all the state transitions in the *Generated Program*, and the probabilities of selecting the initial state and the final state is sufficiently close to one.

(3) The product of corresponding probabilities of all the operation code state and the address correspondences in the *Generated Program* is sufficiently close to one.

1. 8. *Analyzer*

Analyzer reaches the element M_i in *Memory* whose pattern field differs from the *Unknown Problem* (UP) only in one ⟨term⟩ operand. If there is such element M_i, he can extract one common element and two different elements as follows. The pattern field of the common element is that of UP or M_i whose different ⟨term⟩ operand is replaced by TERM pseudo operand. Then, the definition field of the element is the common part of the definition fields of UP and M_i. The pattern fields of the two different elements are the different ⟨term⟩ operands themselves, and the definition field of each different element is the difference between the definition field of UP or M_i and that of the common element.

2. *Learning procedure*

In this section we will explain in detail the learnıng procedure of each component in the *Learning Body*. As the learning procedures of STM and OACM are almost similar and the learning procedure of ISFV and FSFV are also almost similar, we will mention the learning procedure of STM and ISFV only. In the following description we denote the length by L and the state transition sequence of a Generated Program by

$$S_{t_1} \to S_{t_2} \to \cdots\cdots \to S_{t_L}.$$

2.1. *Learning procedure of STM*

When a Generated Program has been judged as *Success* (or *Subsuccess*), learning of the following type is performed.

$$\begin{aligned} a_{t_i t_{i+1}} &\to a_{t_i t_{i+1}} + b_{t_i} \cdot k_S \\ b_{t_i} &\to b_{t_i} + b_{t_i} \cdot k_S \\ (i &= 1, 2, \cdots\cdots, L-1) \end{aligned} \tag{2}$$

where k_S is a positive constant, which will be discussed later. Thus the transition probabilities will be changed as

$$\begin{aligned} p_{t_i t_{i+1}} &\to \alpha p_{t_i t_{i+1}} + (1-\alpha) \\ p_{t_i j} &\to \alpha p_{t_i j} \\ (i = 1, 2, \cdots\cdots, L-1 \quad & j = 1, 2, \cdots\cdots, n \neq t_{i+1}) \end{aligned} \tag{3}$$

where

$$\alpha = \frac{1}{1+k_S} \tag{4}$$

Thus, a linear operator has been operated. Though there are various operators for changing probabilities, the linear operator seems to be the most useful one for our experiment because they promise effective learning and the analysis concerned is easy. In a matrix form, we employ the following BOSH-MOSTELLER's stochastic learning model[1] for our linear operator.

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

where, $\boldsymbol{a}$ is a positive number, $\mathbf{I}$ is the $n \times n$ identity matrix and $\mathbf{\Lambda}$ is the following $n \times n$ matrix.

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \lambda_1 \cdots\cdots \lambda_1 \\ \lambda_2 & \lambda_2 \cdots\cdots \lambda_2 \\ \cdots & \cdots\cdots\cdots \\ \lambda_n & \lambda_n \cdots\cdots \lambda_n \end{pmatrix}$$

$$0 \leq \lambda_k \leq 1 \qquad \sum_{k=1}^{n} \lambda_k = 1.$$

When we apply $\mathbf{T}$ to the probability vector $\mathbf{P}$, we obtain

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

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

Then, each component of $\mathbf{P}$, is changed as follows.

$$\begin{aligned} p_i &\to a p_i + (1-a)\lambda_i \\ (i &= 1, 2, \cdots\cdots, n). \end{aligned} \tag{7}$$

After we apply $\mathbf{T}$ to $\mathbf{P}$ N times repeatedly, we obtain

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

Thus we know $\mathbf{T}^N\mathbf{P}$ tends to $\boldsymbol{\lambda}$ if $0 < a < 1$ as N becomes large, i.e. the i-th component of $\mathbf{T}^N\mathbf{P}$ tends to λ_i.

In case of (3), λ_i equals to one. When a *Generated Program* has been judged as *Failure*, the learning of the following type is performed.

$$
\begin{aligned}
&a_{t_i t_{i+1}} \rightarrow a_{t_i t_{i+1}} - b_{t_i k_f} \\
&\quad b_{t_i} \rightarrow b_{t_i} - b_{t_i k_f} \\
&\quad (i=1, 2, \cdots\cdots, L-1)
\end{aligned}
\tag{9}
$$

where $0<k_f<1$. But, if $a_{t_i t_{i+1}} - b_{t_i k_f} \leqq 0$, $a_{t_i t_{i+1}}$ and b_{t_i} are left unchanged.

Thus the transition probability which is larger than k_f will be changed as follows.

$$
\begin{aligned}
&p_{t_i t_{i+1}} \rightarrow \beta p_{t_i t_{i+1}} + (1-\beta) \\
&\quad p_{t_i j} \rightarrow \beta p_{t_i j} \\
&\quad (i=1, 2, \cdots\cdots, L-1 \quad j=1, 2, \cdots\cdots, n \neq t_{i+1}),
\end{aligned}
\tag{10}
$$

where

$$\beta = \frac{1}{1-k_f}.$$

The criterion that a *Generated Program* is judged as *Success* (*Subsuccess*) will be equivalent to the fact that the state transition sequence corresponding the *Generated Program* contains the state transition sequence $\bar{S}_{jk}$ which begins with state S_j and ends with state S_k, passing through an ordered state sequence (the probability of which is denoted by $\bar{p}_{jk}$). The convergence of $\bar{p}_{jk}$ can be proved if the effect of failure punishment is negligibly small (the proof is shown in [4]). In other words $\bar{p}_{jk}$ tends to unity after a large number of the program generating trials.

In the previous discussion we presumed k_S is a constant. However, it will be better to make the value of k_S depend on the length L of a *Generated Program* judged as *Success* (*Subsuccess*), because, if we assume the minimum length of the solution program to be L_m, $L_m - 1/L-1$ transitions of all, the L transitions satisfy the following relationship, i. e.

$$1-\alpha = k_S' \frac{L_m - 1}{L-1}, \quad \text{where} \quad 0<k_S' \leq 1. \tag{11}$$

Since L is usually unknown, the minimum value 2 may be used. Then, obtain

$$k_S = \frac{k_S'}{L-1-k_S'} \qquad \alpha = 1 - \frac{k_S'}{L-1}. \tag{12}$$

Smaller value of k_S' may be desirable from the learning speed point of view, but the smaller k_S' the more redundant program will be experienced. Such situations were confirmed in the experiments shown in Table 2.

Table 2. The learning results of MINUS (TERM).

k_S'	α	$\bar{N} = \frac{1}{20}\sum_{i=1}^{20} N_i$	$\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(N_i - \bar{N})^2}$	$\bar{T} = \frac{1}{20}\sum_{i=1}^{20} T_i$	$\bar{E} = \frac{1}{20}\sum_{i=1}^{20} E_i$
0. 8	0. 8	965	189	2. 2 min.	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. 675	1401	155	2. 9	0/20

Note N_i: number of program generation trials required complete learning with the i-th experiment
T_i: time required to complete learning with the i-th experiment
E_i: number of useless states included in the lǝarned program with the i-th experiment

2.2. *Learning procedure of ISFV*

Only when a *Success* (or *Subsuccess*) *Program* whose length is equal to or smaller than the minimum length of the post *Success* programs have been generated, ISFV performs the learning as follows.

$$a_{0t_1} \to 1, \quad a_{0t_i} \to \prod_{j=1}^{i-1} (1-p_{t_i t_{j+1}}) \qquad (13)$$
$$b_0 \to 1 + \sum_{i=2}^{L} \prod_{j=1}^{i-1} (1-p_{t_j t_{j+1}}).$$

Thus the probability to select S_{t_i} as an initial state is changed as follows.

$$p_{0t_i} = p_{0t_{-1}}(1-p_{t_{i-1}t_i}) = p_{0t_1} \prod_{j=1}^{i-1} (1-_{t_j t_{j+1}}) \qquad (14)$$
$$p_{0t_1} = 1 \Big/ 1 + \sum_{i=1}^{L} \prod_{j=1}^{i-1} (1-p_{t_j t_{j+1}}).$$

Since the essential part of the state transition sequence of *Success* (*Subsuccess*) *Program* is $\bar{S}_{jk}$, the problem is to search for the true S_j among S_{t_i}'s. By such a learning procedure, the probability of taking S_{t_i} as S_j becomes smaller, as the transition probability of $S_{t_{i-1}} \to S_{t_i}$ grows larger. We can prove the convergence of this learning procedure (the proof is shown in [4]). That is, the initial state will be uniquely determined finally.

2.3. *Conclusion of the learning procedure*

The probability with which *Generator* generates a *Success* (*Subsuccess*) program is given by

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

where (as to the following notation refer to [5]).

$h_{ij}^{(k)}$: the probability that the generating process, starting from state S_i, will ever reach state S_j without passing state S_k.

b_{ij}: the probability that the generating process, starting from state S_i, goes to state S_j for the first time.

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

$\mathbf{B}_i = (b_{ij}) = \mathbf{N}_j \cdot \mathbf{R}_j$

$\mathbf{N}_k = (\mathbf{I} - \mathbf{Q}_k)^{-1}$

$\mathbf{Q}_k$: $(n-1) \times (n-1)$ matrix deleting the kth row and the kth column from $\mathbf{P}$.

$\mathbf{N}_{kdg}$: diagonal matrix having the diagonal elements of $\mathbf{N}_k$.

$\mathbf{R}_j$: $n-1$ component columh vector which is made by deleting jth component from the jth column vector of $\mathbf{P}$.

$\mathbf{I}$: unit matrix.

From the convergence of $\bar{p}_{jk}$ and the learning of ISFV and FSFV, it will be obvious that p_{SUC} tends to unity after sufficiently many trials. Further, by considering the convergence of OACM we reach the following conclusion concerning our learning procedure.

By our learning procedure, the solution program will be certainly learned in the long run. The program obtained by such learning will not be always the best one, that is, the

```
   MPROG

CAD    N
ADD    KLM
STR    WORK1
CSB    H1
SUB    H2
STR    WORK2
LINK   GPROG
STR    WORK3
CSB    WORK3
SUB    WORK2
ADD    IJ
SUB    WORK1
STR    ANSWER
HALT

   UNKNOWN

DIV(ABCD,EFG)

DIV(ATOM,ATOM)

   ADVICE
```

Responce from the computer to the *Problem-P 2.* The computer expresses the status of the unknown function DIV and requests the advices required.

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

Give the *Problem-P 3* to learn the unknown funtion DIV (ATOM, ATOM).

```
   MPROG

LINK   GPROG
STR    C
HALT

   UNKNOWN

DIV(A,B)

DIV(ATOM,ATOM)

   ADVICE
```

Responce for the *Problem-P 3.*

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

Give the advice to solve the *Problem-P 3* with numerical informttions: SET (A=2. 449048, B=1. 414) GOAL (C=1. 732 and A=2. 449048 and B=1. 414)

```
   LEARNED
```

Learning for the *Problem-P 3* has been completed in about 24 seconds.

```
MOVE(SUB(ADD(SUB(MINUS(DIV(ABCD,EFG)),SUB(MINUS(H1),H2)),IJ),
ADD(KLM,N)),ANSWER)
```

Give the *Problem-P 2* once more.

```
   MPROG

CAD    N
ADD    KLM
STR    WORK1
CSB    H1
SUB    H2
STR    WORK2
CAD    ABCD
DIV    EFG
STR    WORK3
CSB    WORK3
SUB    WORK2
ADD    IJ
SUB    WORK1
STR    ANSWER
HALT
```

The correct program was printed as a solution of the *Problem-P 2* in about 2. 5 seconds.

Note. GPROG denotes *Generated Program*

Fig. 2. An example of process of gaining the solution of *Problem-P 2.*

```
   MPROG

CAD    N
ADD    KLM
STR    WORK1
CSB    H1
SUB    H2
STR    WORK2
CAD    EFG
ADD    ABCD
STR    WORK3
CSB    WORK3
SUB    WORK2
ADD    IJ
SUB    WORK1
STR    ANSWER
HALT
```

Note. WORK i (i=1, 2, 3) denotes working area address.

Fig. 1. Massage for *Problem-pl* (solution).

length of the program may not be the minimum among the possible solution programs. However, when we choose the parameters of the learning procesure properly it will be possible to obtain the best program at a sacrifice of the learning speed.

3. *Examples of the experiment*

First we educated the B-computer to master 5 functions, MOVE, ADD, SUB, MINUS and PLUS, spending about 65 minutes by NEAC-2206, in which both the printing time and the manual operation time are included. After that we showed the *Problem-P*1,

MOVE(SUB(ADD(SUB(MINUS(ADD(ABCD, PLUS(EFG))),
SUB(MINUS(H1), H2)), IJ), ADD(KLM, N)), ANSWER),

then the B-computer printed out a program (solution) shown in Fig. 1 after about 1.5 seconds. However when we showed the *Problem-P*2 which included unknown function DIV,

MOVE(SUB(ADD(SUB(MINUS(DIV(ABCD, EFG)), SUB(MINUS(H1),
(H2)), IJ), ADD(KLM, N)), ANSWER)

then the B-computer at once printed a message shown in Fig. 2 and halted requesting *Advice*. In Fig. 2 shows a sequence of messages printed out or typed in, by the time the solution program of the *Problem-P*2 was obtained.

4. *Conclusion*

The system reported in this paper may be viewed as a compiler generating system in a sense, that is, in this system the B-computer learns the semantics of the syntax of *Problem* expressed in the language of the H-computer and grows up to a kind of a syntax directed compiler[6]. In order to extend our system more powerful, inclusion of a time dependent learning procedure, will bo necessary, because, we can determine the state sequence uniquely by considering the time. Further, there are many problems still remained such as extending the *Problems* themselves and making the *Advice* of higher level. For the purpose of realizing an artificial intelligence, the language of the H-computer itself will be the object of further research. Anyway, we believe our self-programming system is a very meaningful one from our experiences, as an example of the artificial intelligence.

References

[1] Bush, R. R. and F. Mosteller, Stochastic Models for Learning. John Wiley & Sons, Inc., New York N. Y., 1955.

[2] Simon, H. A., Experiments with a Heuristic Compiler. *J. ACM 10*, 4.

[3] Sclesinger, S. and L. Sashkin, POSE: A language for Posing Problems to a Computer. *C. ACM 10*, 5.

[4] IShida, T., H. Yasui, H. Sugiyama, and K. Joh, An Experiment with a Self-programming System. *Joho Shori*, *8*, 3.

[5] Kemeny, J. G. and J. L. Snell, Finite Markov Chains, D. Van Nostrand Company, Inc., Princeton, New Jersey, 1960.

[6] Irons, E. T., The structure and Use of the Syntax Directed Compiler. In Third Annual Review of Automatic Programming, Pergamon, New York, 1963.