R. M. Friedberg
B. Dunham
J. H. North

A Learning Machine: Part II*

Abstract: An effort is made to improve the performance of the learning machine described in Part I, and the over-all effect of various changes is considered. Comparative runs by machines without the scoring mechanism indicate that the grading of individual instructions can aid in the learning process. A related study is made in which automatic debugging of programs is taken as a special case of machine search. The ability to partition problems and to deal with parts in order of difficulty proves helpful.

Introduction

The experiment described in Part I was continued after an interruption of several months. Two immediate objectives were set: (1) an explicit measure of Herman's learning efficiency, and (2) a better understanding of the factors which govern that efficiency. We are interested in Herman because elements which help or hinder his small-scale performance can well influence more substantial learning machines.

But how is efficiency to be measured? Suppose changes are made in Herman's program randomly, without the benefit of success numbers. For any reasonable problem, a correct program will be hit upon eventually. The question is *how much faster* Herman, with the aid of the scoring mechanism, will develop correct programs than he would merely by random, trial-and-error search.

Not all random searches are alike. Suppose we discover two men on a lake: Samson and Homer. Each is blind and cannot fix his precise location. Nevertheless, by dropping a lead line, each can determine exact depths. Samson is somewhat stolid. He is blown by chance winds about the lake, but he finds pleasure in dropping his lead line every five seconds and recording the depth. Since one drop follows so immediately upon another, his successive positions are close together and do not generally differ much in depth. Homer, on the other hand, takes great pleasure in surprise. He too is bandied about by chance winds, but he waits a full hour between drops. In this way, his successive positions tend to be rather remote; and he has no idea what will come next.

Suppose Samson and Homer compare the first 10,000 depths each has recorded. If we assume the lake remained unchanged, there is no reason why either man should have scored more *deep* depths than the other. Samson, of course, obtains his extreme readings somewhat in bunches. Suppose now we penalize Samson for his stolidity. Every time he records a deep depth of a certain magnitude, he must wait an hour before his next drop. His percentage of deep drops will become smaller, since he cannot take full advantage from having reached a deep section of the lake.

With this fantasy in mind, we set up two different random machines. One was like the penalized Samson, and the other was like Homer. When the Samson machine failed, only one or two of the 64 active instructions were changed. When a successful program emerged, the whole machine was started from scratch. The Homer machine, on the other hand, underwent total revision after each failure. As one would expect, Homer far surpassed Samson in the average speed with which he obtained correct programs. Nevertheless, Samson serves as a more valid basis for appraising Herman's success-number mechanism, because Samson is almost exactly like Herman except for lacking such a mechanism.

Teddy

Before we attempted to measure Herman's learning efficiency, we made two changes in his mode of operation by *priming* and *reset*. Both were calculated to improve his performance. In this way, production runs on the IBM 704 could be cut down and very much better statistics obtained.

As noted in Part I, a successful program is likely to contain *op* 0 in the initial location and *op* 1 or 3 in the final location, when the two locations are used for input-output purposes. We *prime* Herman by guaranteeing

*"A Learning Machine: Part I" by R. M. Friedberg, appeared in the *IBM Journal*, **2**, No. 1, 2-13 (January 1958).

that this is the case. Priming is thus an *ad hoc* adjustment to Herman's particular characteristics; *reset,* on the other hand, is more fundamental. Although the mechanism of success numbers enables us to "criticize" and modify the instructions in the various locations, it in no way affects the data bits, which carry over unchanged from one run to another. This means there is a certain "dark area" in the experiment, in that a part of the machinery which most influences the outcome of a problem is almost totally independent of the success-number bookkeeping. Further, an important element which could easily be kept constant from run to run, thereby reducing the over-all complexity of the situation, is permitted to vary. Herman is *reset,* therefore, after every run by inserting ZERO in all data locations which do not represent the selected inputs. From here onward, we shall assume machines to be primed and reset, unless otherwise specified.

The efficiency with which variant machines dealt with Problem 1 was used as a rough touchstone of their learning potential. In this problem, D_0 is the input location, D_{63} is the output location, and the criterion of success is that the output bit should be identical to the input bit. Table 1 shows the general effect of priming and reset. At least 2,000,000 trial runs were made to obtain the statistics for each of the four possibilities.

Table 1 **Average number of trials required to achieve a perfect program for Problem 1.**

	Reset	*No Reset*
Herman primed	15,197	57,281
Herman unprimed	78,829	477,019

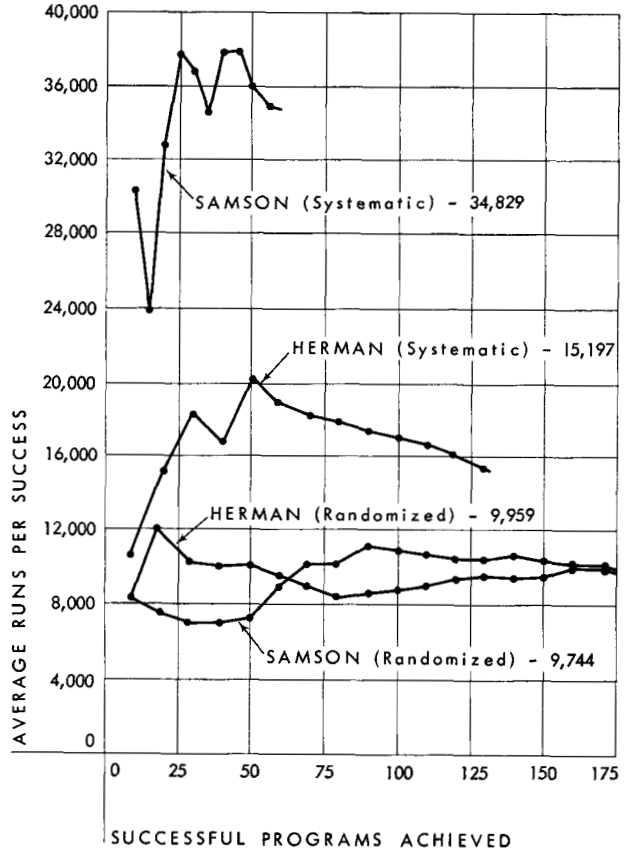
Herman's performance on Problem 1 was then compared to that of Samson and Homer, mentioned earlier. Samson was so set up that one active instruction was replaced by its inactive counterpart after every failure. After every 64 failures, one of the 128 instructions, chosen randomly, was replaced by a new random one. The interchange of active and inactive instructions was done both systematically and by random choice. The latter mode of operation is more efficient, since there is less likelihood that the machine will *cycle.* When changes are made in systematic order ($I_0, I_1, \ldots, I_{63}, I_0, I_1, \ldots$), almost duplicate programs will occur every 128 failures. Hence, programs that have already proved failures may be run again, which will inhibit the speed of learning. Samson was, therefore, set up in both ways for Problem 1. Both machines were given the usual 2,000,000 trials. On the average, the "systematic" Samson, which most resembled Herman, required 34,829 runs to achieve success. The "randomized" Samson required 9744. Homer was then tried. Set up in the manner earlier described, he achieved over 1000 perfect programs at an average of 356 trial runs.

The difference between the two Samsons suggested a possible change in Herman. "Criticism" of individual locations after failure might no longer be made in systematic order, but randomly. The "randomized" Herman was in fact set up, and (2,000,000 trials, Problem 1) averaged 9959 runs per success. Figure 1 shows graphically the comparative performance of the two Samsons and Hermans. The fact that the "randomized" Herman does not maintain the supremacy over the corresponding Samson which was shown by the "systematic" Herman is not surprising. Because random success numbers are assigned at the start, a problem must be run for some time before the success mechanism can take full effect. If average runs are sufficiently short, the success mechanism can in fact inhibit performance, since, in the initial stages, instructions with the lowest success numbers may well be ones which have had no effect whatever on the problem.

Because it was felt that Herman's performance could be substantially improved, it was decided to modify the general scoring mechanism; and a new machine, Teddy, was put together.

Teddy differs from Herman only in the way the associated Learner functions. Two general motivations governed his design: (1) elimination of *dark areas,* and (2) reduction of *traffic jams. Dark areas* arise when the critical mechanism does not attack those elements which have in fact had most to do with past performance. It

Figure 1 **Record of performance on Problem 1, random input.**



may either punish and reward innocent bystanders, or leave unnoticed true heroes and villains. *Traffic jams* arise when the machine gets itself into a difficult situation from which it can emerge only very slowly, if at all. For example, on a given production run where Herman learned Problem 1 45 times, in 42 cases it never took him more than 30,000 trials. In the remaining three cases, he did not take less than 86,000.

What then are the basic differences between Teddy's scoring mechanism and Herman's? *First, success numbers are modified only for participants.* A *participant* is an active instruction which was either executed in the run in question or referred to by the *b* part of an executed *op* 3. *Second, success numbers are both raised and lowered.* After every success the appropriate success numbers are increased by two. After every failure, the appropriate success numbers are reduced by two. *Third, the method of assigning success numbers is changed.* Upon initialization, all of the instructions are assigned a success number of 1000. The S_i of an instruction later introduced is the mean of the current success numbers of the other instructions. *Fourth, a new system for handling maximum and minimum numbers is introduced.* No number is taken as S_m, and scaling is eliminated. An instruction whose success number drops to 256 is replaced by a new random instruction. *Fifth, there is insured modification of the program after failure.* Only the participants are subjected to "criticism," and this process continues until one of the inactive instructions becomes active.

The reasons for these changes are not particularly subtle. Restricting "criticism" and success-number modification to participants is designed to eliminate dark areas. Lowering of success numbers upon failure and the new treatment of S_i and S_m are designed to remove the tendency of success numbers, after a run of a certain length, to bunch together just below S_m. The latter makes the interchange of active and inactive instructions almost automatic (since the size of the two numbers will be great relative to their difference), and leads also to traffic jams. Suppose, for example, a small number of Herman's instructions have success numbers substantially lower than the bunch at the top. These few may receive the great burden of the Learner's "criticism," even though they may not have been participants for some time. As a result, Herman may work himself out of a jam quite slowly. Indeed, the longer a problem runs, the greater the tendency of success numbers to bunch at the top. Thus, the machine (for a variety of reasons) may become more-or-less static in its behavior, its basic operations being largely independent of the changes made when it fails. As a manifestation of this phenomenon, an unprimed, unreset Herman was made to arrive at and retain for some time a program merely to finish, even though given an automatic failure nine times in ten.

Because Teddy's over-all design is so much a function of earlier experience with Herman, it may be of some interest to set forth a few details of the result just mentioned. In a given lesson, we presented Herman (unprimed and unreset) with a new problem in which the only criterion of success is that the program finish in time. The Teacher, however, was made to report only one of every ten true successes to the Learner as a success, and the other nine as failures. The frequency of true success rose gradually from a small initial percentage to almost 100 per cent. It then remained well above 90 per cent for better than 400,000 trials, during which a good many of the 128 locations were affected by random changes. We examined several programs that arose during the latter 280,000 trials. They all lacked any *op* 0 instructions in the last ten pairs of active and inactive instructions, and possessed a number of *op* 0 instructions elsewhere with addresses designating these last locations. In fact, the empirical check indicated that the last 11 pairs of instructions were not replaced by any new random instruction in the final 100,000 runs. Both of these features are obviously prone to favor a success for the problem in question without necessarily ensuring it. When Samson (unprimed and unreset) was substituted for Herman in this experiment, the frequency of true successes did not rise above 15 per cent. Thus, it can be seen that the success-number mechanism is very effective for this problem, both in developing a high frequency of success and in maintaining it despite random changes in the program. This contrasts with the inability of Herman to maintain an almost successful program for Problem 1 when given a systematic failure only one true success in ten (Part I, Experiment 9).

Teddy's record of performance is somewhat better than Herman's. In a total run of 500,000, he was able to solve Problem 1 after an average of 1360 programs tried. It should be noted, however, that the way in which the problem is posed by the Teacher and the method of counting runs are both different from that described in Part I. Since Teddy is reset after every run, he will obtain an identical output given an identical input, provided no instructions have been changed. Hence, it is simpler to know when he has arrived at a perfect program—he need only have tried all of the possible input circumstances and obtained correct outputs. With this in mind, we set up the following procedure: a given program tries out all the input conditions before a change is made. Success numbers are appropriately modified, and one set of "criticisms" is made per failure. The resulting program is then tried, and so on. In tabulating the result, it is more convenient to count the programs tried than the individual runs. With this new mode of bookkeeping in effect, Herman (primed, reset, "randomized") required on the average 2890 program trials to solve Problem 1. The "randomized" Samson needed 4603 trials, and Homer, 321. Figure 2 shows these and other results.

Teddy and the "randomized" Herman were also tried on a few two-variable problems, in which D_0 and D_5 were taken as input locations, and D_{63} as the output. Table 2 provides a record of their achievement. The functions indicated are the familiar truth-functional ones.

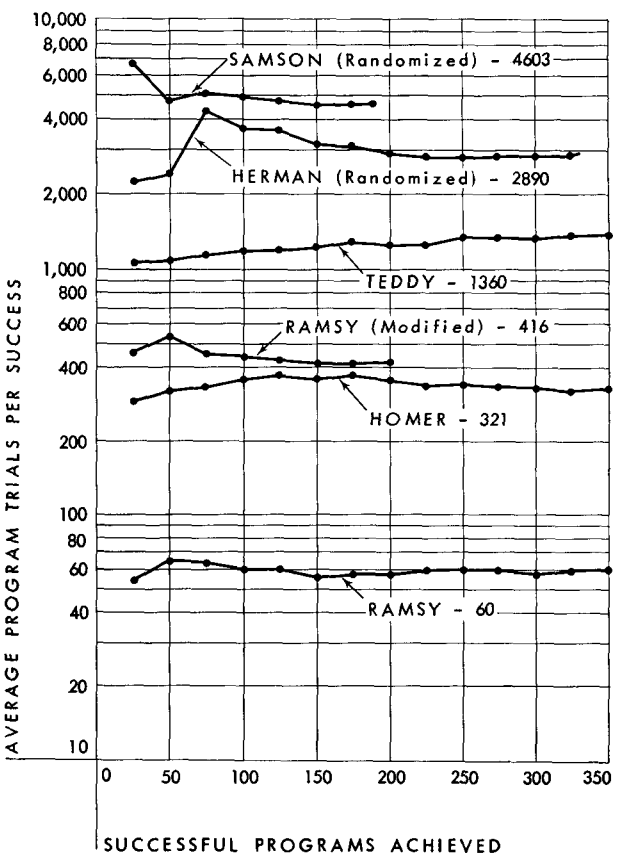
Table 2 **Comparative performance of Teddy and Herman (randomized) on two-variable problems.**

Function	*Average programs tried before success and number of successes obtained*			
	Teddy		*Herman*	
AND	24,896	15	225,508	3
INCLUSIVE-OR	44,539	9	97,978	6
NOT-IF-THEN	18,633	8	141,306	3

One of the major objectives of Part I was to determine whether the grading of individual instructions would aid in the learning process. Samson, Herman, and Teddy, in their various forms, are all inhibited in that changes are made more or less one at a time. The generally superior performance of those machines with a success-number mechanism over those without does serve to indicate that such a mechanism can provide a sound basis for constructing a learning machine.

Another aim of Part I was to render more explicit just how a machine, in a progressive sequence of operations, could discover order in the midst of apparent chaos. As we have seen, dark areas and traffic jams are likely to occur and do have a definite effect on the efficiency of the machine. Hence, in setting up future learning machines, we need to consider how these problems are to be managed in the given case. There is, of course, the fascinating prospect that a learning machine might so adjust itself as to eliminate traffic jams and dark areas progressively, as it gains experience.

Figure 2 **Record of performance on Problem 1, systematic input.**



Finally, there is the problem that making changes one at a time can very much inhibit a learning machine. Homer far outstrips Teddy in performance. The particular way in which success numbers function for the machines we have considered makes it difficult to avoid this inhibition. Hence, the question was next raised how an elementary learning machine, which followed reasonably simple but general directions, might be set up without the use of success numbers.

Ramsy

It was remarked in Part I that, although a learning machine might indeed "learn to perform a task without being told precisely how to perform it, it would still have to be told precisely how to learn." For the machines we have been considering, the ability to arrive at a program for solving problems simply from seeing whether trial runs succeed or fail depends in large part on having some effective way of selecting one imperfect program over another. Naturally, it is easy to recognize a perfect program when one comes along; and one might, as Homer does, simply try out one new program after another. Still, this method, though suggestive, does not seem very promising. The harder the problem, the less likely it will help us. If we could arrive at programs not yet perfect which have, nevertheless, a certain figure of merit in their favor; and, if we could use such programs as a decent basis for obtaining new programs with a higher figure of merit, perhaps we might develop a reasonable learning machine after all. But how is all this to be done?

Suppose we simplify the problem somewhat. Let us say we want a *self-debugging* machine capable of making an efficient, progressive search for correct programs, starting from scratch in each case. What exactly would such a search be like? Let us go back to the lake-bottom fantasy described in the Introduction. We now postulate Thales, a third man. Thales is neither blind nor blown about by chance winds. He can always return to a spot just left. To determine depths, he too must drop a lead line; but he does have the capacity to try out nearby positions before moving on to them. In this way, he can always move to a deeper part of the lake until he reaches a position of desired depth unless, of course, he is stopped at some point which is deep relative to its immediate surroundings but not to the lake as a whole. Under the latter circumstances, he must accept new positions which are not necessarily better, in order to get out of the immediate dead end.

Now, although the "Thales" technique of directed search may seem simple and straightforward, we do not

in fact understand it or its applications fully. Nevertheless, we do know that it is a powerful method, which we have already used with considerable success on a number of occasions where other problem-solving techniques failed. It is typically applied to problems with answers easy to recognize but difficult to calculate. Hence, the analogy to the machine-learning experiment is helpful. On the one hand, we may receive some hints as to how the learning machine can be set up more efficiently. On the other hand, we are provided with an additional motivation for the experiment. A learning machine based on the principle of directed machine search would provide a rather illuminating example of this technique. After all, our basic objective in building and studying computers is to obtain greater problem-solving capacity. For the latter, we need not only better machines, but also a better understanding of how to use them.

Most problems we encounter can be broken down into parts, and often these parts are not difficult in themselves. All of us have dealt with outwardly hard problems which became easy when reduced to a set of simpler subproblems. Suppose two imperfect methods for solving a problem are at hand. One does no good at all. The other manages a certain segment of the problem. We would obviously attach a higher figure of merit to the second. If there were some way of leaving undisturbed those features of the better method which contributed to its partial success, while modifying other features which led to its partial failure, we should also have a decent basis for obtaining a new method with an even higher figure of merit. In this way, we could proceed to the solution by a sequence of definite steps. Our scheme of operation would then resemble Thales, in that a proposed step would be accepted only when in the right direction. Suppose the various parts, however, though much easier than the total problem itself, are not alike in difficulty. If the solution of one part does not help that of another, it would be advantageous to attack the more difficult parts first. Since we hope to leave undisturbed those features of our method which contribute to whatever partial success has been obtained, the farther on we get with a problem, the less of the method we have available to modify.

The machine Ramsy is based primarily upon two edicts: (1) partition the problem into parts, and (2) deal with the more difficult parts first. An added principle of operation is derived from Homer's superiority over the penalized Samson. When a purely random search is made, wholesale eradication should follow failure. Needless to say, dark areas and traffic jams are to be avoided.

It is convenient to retain the three basic blocks of the learning machines already discussed, but to modify their mode of interaction. In Part I, the machine was broken down into Teacher, Learner, and Slave (that is, Herman). We sometimes use the name of the Slave loosely in describing the whole machine, but no confusion should result from this. To this family of three, we provided certain information, namely the possible inputs and related outputs of the problem in question. No other information was given. Specifically, we did not adjust the family externally from problem to problem. In setting up a new family machine, we wish the same rules to apply for posing problems.

The Slave Ramsy is identical to Teddy and Herman in kinds of instruction and number of locations. The associated Teacher and Learner, however, have somewhat reversed roles. The Teacher is more active, the Learner more passive, and more information passes between them. The partitioning and ordering of the problem are done by the Teacher. The Learner keeps a limited set of records and provides a random-number generator. There are no success numbers or inactive instructions.

How is a problem partitioned? If we restrict our attention to "bit-manipulation" problems (which seem sufficient to our purpose), partitioning is straightforward. Suppose we wish Ramsy to solve the familiar problem AND for two variables, the input and output locations being D_0, D_5, and D_{63} as before. There are four possible input cases (two ONES, two ZEROS, *et cetera*), and these define the four parts of the problem.

How are the parts to be ordered? Assume that Ramsy is primed with an *op* 1 or 3 in the output location and will retain that operation. With *op* 1, it is easier to produce ZEROS than ONES; with *op* 3, the reverse. If a problem produces more ZERO outputs than ONES, it is better to have *op* 1 in the appropriate location; otherwise, *op* 3.

The Ramsy family's mode of operation can now be briefly summarized. A problem is posed as always. The Teacher primes the Slave appropriately and orders the parts in terms of difficulty. A most difficult part is given the Slave to solve first. If he fails on a given run, all of the participating instructions (with the exception of the bits determining the primed operations) are replaced by new random instructions. If he succeeds, all of the participants become *bound;* and the next part of the problem is undertaken. Bound instructions are exempted from later eradications unless Ramsy becomes *stuck,* which occurs when no participants can be replaced after a failing run because all are bound. Under such conditions, Ramsy is filled with new instructions and the problem begun again.

Ramsy proved far more successful than Homer. On Problem 1, for example, he required, on the average, only 60 trial programs in obtaining 500 solutions. Figure 2 shows graphically his performance. This superiority was also maintained on a number of two-variable problems, as indicated in Table 3.

To test the efficiency of the Homer-Samson principle, some limited runs were also made on a modified Ramsy in which never more than one participant was replaced by a random instruction. As expected, the more stolid machine proved less efficient. On Problem 1, for example, the modified Ramsy generated 200 perfect programs at an average of 416 trials each, as shown in Figure 2. A brief run was also made in which Ramsy was presented with the easier of the two problem conditions first. He averaged 144 program trials to solve Problem 1.

Table 3 **Comparative performance of Ramsy and Homer on two-variable problems.**

Function	*Average programs tried before success and number of successes obtained*			
	Ramsy		*Homer*	
EXCLUSIVE-OR	12,934	31	199,910	0
AND	1758	32	19,445	5
INCLUSIVE-OR	216	60	7208	42
NOT-IF-THEN	6820	35	14,247	16

Another variant of Ramsy included a built-in mechanism for shortening programs. This was done by a rather simple system of address modification which served to eliminate redundant participants. The technique worked smoothly but was of only moderate interest to the over-all experiment. Programs were very much shortened, but the speed of learning was not affected.

Still another variant included an elementary method for making D_{63} a function of D_5, when the latter served as an input. Learning efficiency was increased, but it was not easy to generalize the technique to deal with more involved cases.

The chief drawback to Ramsy is, of course, his growing tendency to become stuck as more and more complicated problems are encountered. Undoubtedly, an improved mechanism can be devised for dealing with this situation. The problem is analogous to that of Thales when he reaches a depth which is maximum for a given area, but not yet deep enough. Without taking a totally fresh start, he must "back-up" slightly in order to get out of the immediate dead end. Traffic-jam techniques appropriate for Thales may prove workable for Ramsy.

Added experience has revealed, however, an additional drawback to Ramsy, which would seem to make it profitable to start over with an entirely new Slave. Since the basic method of learning is quite different from that introduced in Part I, the particular operations selected for Herman need not be retained. These require, on the average, 260 microseconds to execute. If the tiny computer which we simulate on the large machine had an order code more closely akin to some of the actual instructions already available, the machine time required to carry out the experiment could be substantially reduced. For a variety of reasons, this would very much facilitate the conduct of the experiment.

Conclusion

In setting up an experiment in which we study the properties of small, easily controlled machines as a guide to larger machines, we face special problems in understanding our results. Those characteristics of the small machine which obtain simply because it *is* small must be differentiated from more fundamental properties which may hold in the larger case. We have seen that the amount of change made when a program fails has a definite bearing upon the average speed with which successful programs are obtained. Homer makes large-scale changes upon failure, and surpasses Samson for this reason. Thales, on the other hand, undertakes only small changes; but those changes made are likely to be in the right direction. Hence there is a definite tie-in between the size of change made and our capacity to compare related positions. The results obtained do indicate that a simple reinforcement of individual instructions can aid in the learning process; but we have not wholly succeeded in setting up a machine based upon this principle. The difficulty is that, although small-scale changes are made upon failure, our scoring mechanism is admittedly loose. The importance of dark areas and traffic jams has also been emphasized. As we have seen, there are special problems in recognizing, avoiding, and getting out of the latter. The results with Ramsy indicate that the ability to partition problems and to deal with the parts in order of difficulty does prove helpful. In this connection, we found it worthwhile to note the analogy between machine learning and the problem-solving technique of directed machine search.

Where we should go from here is not entirely clear. Perhaps the experiment, with a radically different Slave, could be set up in a closer analogy to Thales. We have a somewhat vague but quite persistent sentiment that the methods we have used to bring about learning are too passive. Some scheme of ensured referencing or execution of inputs, such as that briefly suggested near the end of the last section, might well be introduced. In all events, we find the unanswered questions as fascinating as they are difficult.

Received March 24, 1959