PB176 761

CS 50
A.I. Memo #46

SOME SEQUENCE EXTRAPOLATING PROGRAMS: A STUDY OF REPRESENTATION AND MODELING IN INQUIRING SYSTEMS

BY

STAFFAN PERSSON

TECHNICAL REPORT NO. CS50
SEPTEMBER 26, 1966

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Reproduced by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information Springfield Va. 22151

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to Drs. C. West Churchman and Edward A. Feigenbaum for their unstinted time, encouragement, and intellectual inspiration during the work of my dissertation.

I also would like to thank other people--too numerous to list by name--who in various ways have contributed to my completing this thesis.

TABLE OF CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Section

Page

LIST OF ILLUSTRATIONS
(Continued)

Section Page

6. (Continued)

7. EXTRAPOLATION OF SEQUENCIES OF SYMBOLS

1. Introduction

Managerial decision-making traditionally conforms to a separatist philosophy,[1] i.e., decisions are made without questioning the origin of the premises. Thus, accounting-data, engineering-data, sales-data, etc., as supplied by different functional representatives are accepted as valid representations of the state of the decision-environment.

The development of powerful computerized information processing systems has brought an increasingly large portion of managerial functions within reach of mechanization. Usually such mechanization is performed by retaining the basic philosophy of separatism, i.e., given unambiguously represented data, a programmed decision-procedure is employed to attain desired results. This approach, however, does not fully take advantage of the capabilities of modern information-technology, i.e., advantage has not been taken of the possibilities of tailoring the information systems to the needs of the managers. However, the information-needs of the managers are by no means clearly defined. Dr. Harold Koontz in a recent panel discussion[2] described how an information-system tailored to the stated needs of the executives of a progressive company turned out to be a complete failure, because the decision-makers were used to receive information, not to ask for it.

Thus easy access to a large store of raw-data, i.e., largely unedited simple elements of information, does not in itself give an optimal support to efficient decision-making even if it potentially can supply any information needed by an executive. On the contrary, it is a commonly held opinion that efficient problem-solving requires suitable representations of problems, and thus also suitably organized data.

The situation is illustrated in an example due to Morse,[3] where a "typical" inventory-problem such as finding an optimal reordering-policy for appliances is represented as a queuing-problem. The existence of alternate representations of problems implies that the need for data can not be determined until a particular representation is chosen. This suggests that instead of supplying a set of well organized data-representations to an information-processing system, a richer, and more satisfactory situation would be achieved if the "system" could accept a minimal raw-data representation, decide how to represent encountered problems, organize data as required, and then perform the actual data-processing-phase. It may be doubtful if such a system can be designed, however, it is quite possible that the following question at least for certain areas can be answered in the negative.

Are all significant analogies (representations) beyond the capabilities of explicit design-rules, i.e., are all significant analogies "creative"?

The present thesis attempts to show the feasibility of designing an automated system for finding suitable representations in a special case, where

(a) The "raw-data" are restricted to symbol-differentiation.

(b) The problem is restricted to sequence extrapolation.

(c) The number of alternative representations is restricted but expendable.

Although the thesis is restricted to a specific case, the employed methodology and the general structure of the developed system will suggest other areas of application.

The design of the system will proceed via a study of how certain epistemological models of inquiry represent an unlimited store of raw-data, i.e.. "reality", a discussion of the importance of representation in problem solving, to the organization and implementation of a computer-program for finding representations of sequential patterns.

1.1 Inquiry - A Problem of Representation

Although the present thesis attempts to solve a specific representational problem, major parts of the discussion can be held on a general level, thus facilitating future attempts to generalize on achieved results.

We have in the introduction indicated the representational problem of managerial decision-making and its impact on the design of organizational information-processing systems, however, no definition of the problem was given. For the present discussion a definition which abstracts from the situation of particular managers, but still retains the basic representational characteristics is desired. One such definition is:

> The object of managerial decision-making is to initiate correct action in a changing environment.

This definition closely parallels Churchman's pragmatic definition of knowledge:

> "Knowledge is a potential of taking correct action in a changing environment."[4]

It should be noted that by assuming that correct action for the manager is equivalent to correct action for the organization, i.e., the manager is a member of a team in the Marschak-Radner sense, the role of an organization can be defined as:

> Organizations, as its managers, seek a capability of taking correct action in a changing environment.

This means that the objectives of organizations and of managers in the present context can be considered to be synonymous to creation of knowledge, i.e., organizations and managers perform the functions of inquiring-systems.

By studying the concepts of managerial and organizational decision-making at a level of abstraction corresponding to the general concept of inquiry, several advantages can be achieved, e.g.,

a. Experience can be drawn from epistemological models of inquiry.

b. A convenient categorization of "problem-solvers" can be developed.

c. Abstraction from particular functions of managers permits a study of the general problem of representation in inquiry.

Although the ultimate goal should be to establish to what extent the epistemological problem of inquiry can be transformed to a technical problem, the present thesis surveys the general representational problems of inquiry in order to mechanize a particular system. Thus by studying the general problems:

1. What are the necessary functions of an inquiring-system?

and 2. Which functions, if any, of inquiry can be mechanized?

Answers may be found for more specific questions such as:

3. How is the representational function represented in the systems of inquiry?

and 4. How can a particular representational function be mechanized?

The solutions to the problems will have to be sought in several fields such as artificial intelligence, computer science, logic, mathematics, philosophy, psychology, etc.

As will be seen in the discussion, a merging of information from these diverse fields is complicated by linguistic incompatibilities. Thus, the phenomena called "Cartesian Dualism", i.e. the employment of a different language for discourse on cognition than for subjects concerning the material world, will present itself. This clash of languages has been observed particularly in the literature on artificial intelligence, where words such as machine-learning, machine-intelligence, machine-knowledge, etc., have caused a great deal of confusion.[5] However, since these subjects in the future are likely to occur more frequently, the well-known ability of language to adapt itself to new categories through metaphors or genuine creation undoubtedly will eliminate this linguistic block. In the present discussion, however, the incompleteness of language caused by the relative youth of the field of artificial intelligence remains as a problem.

1.1.1 On Models

By a model of a systems we mean any mechanical, chemical, or symbolic representation of its relational structure. A symbolic model consists of a collection of rules, namely:

1. Rules for translation of "reality" (object language) to the language of the model (model-language).

2. Rules for manipulation of sentences in the model-language (syntax of model); and

3. Rules for describing sentences of the model-language in a meta-language (semantics).

Type 1 rules are representations, i.e., reflexive and transitive relations between the sentences of an object-language and the axioms, theorems, and sentences of a model.

The extent to which a model can portray the "object-world" is determined by the richness of the employed languages (Figure 1.1). A "perfect" description requires isomorphism between object and model, a situation achieved easily when the model-language is richer than the object-language. In most practical situations, however, a model is intended to be an easily manipulated abstraction of its object, and therefore the model-language and the object-language are homomorph by design.

To answer the questions set forth in section 1.1 a discussion is necessary at two levels: 1. Inquiry should be discussed in the richest possible model-language. 2. The mechanical design-models of the models of inquiry must be formulated in a language that is simple enough to be translated into a physical realization. The use of two levels of models implies that, a priori, inquiry is not considered to be within reach of purely mechanical processes.[6]

For the purpose of modelling what amounts to thought processes, there is no richer medium than that provided by the human mind augmented by proper tools. As, among others, Craik[7] has realized, "...human thought has a definite function; it provides a convenient small scale model of a process, ...". He also notes that "Again, there is no doubt that we do use external and mechanical symbolizations to assist our own thinking." The external devices are designed, however, by the human mind and we can safely assume that the brain without having to refer to any particular augmentations, is the most powerful medium available for modeling processes.

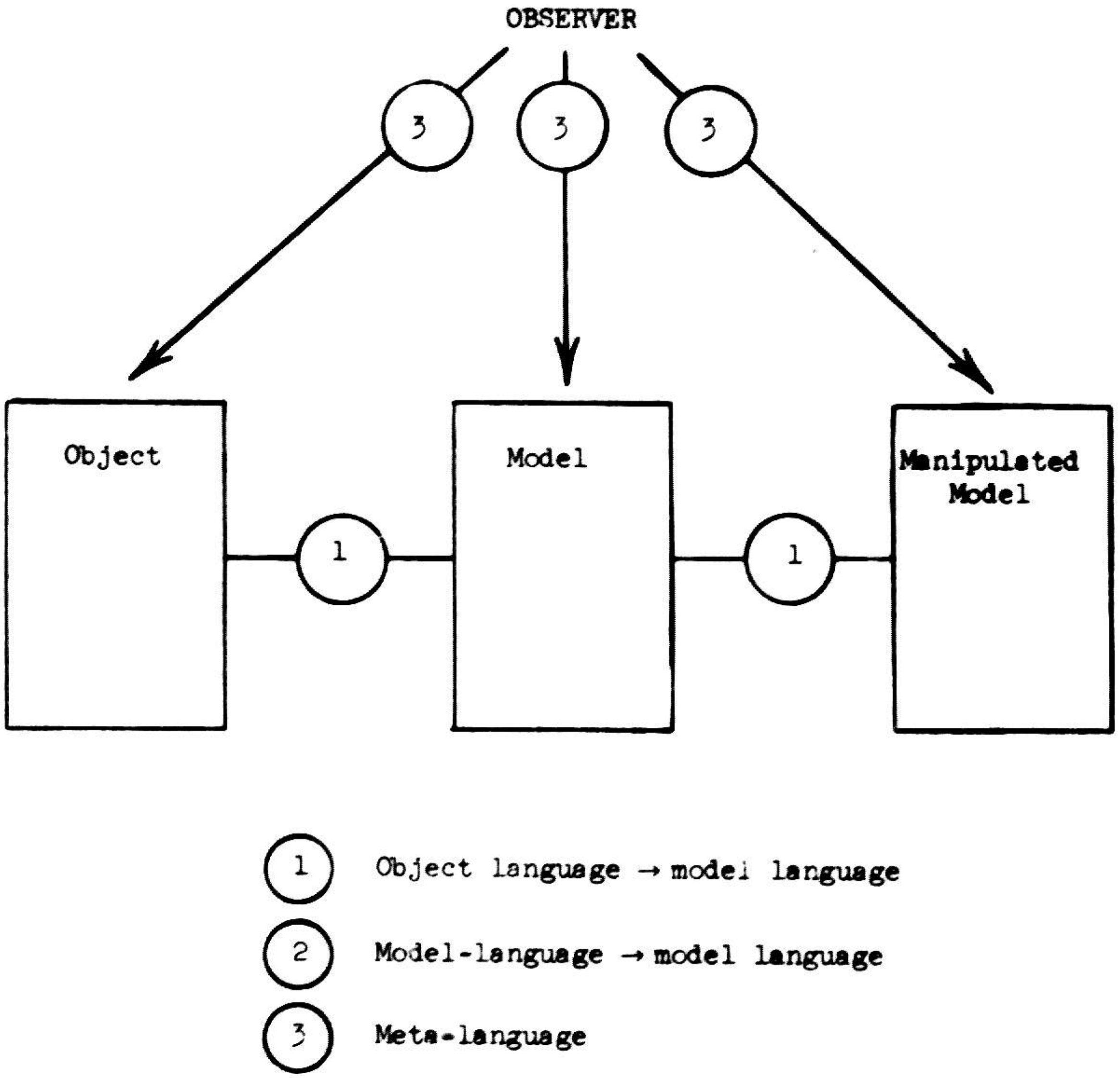

OBSERVER
3
3
3
Object
Model
Manipulated Model
1
1
1 Object language → model language
2 Model-language → model language
3 Meta-language

Figure 1.1

An epistemological model is a model that the mini formulates about itself. Such self-description or self-reproduction is feasible logically[8] and there is no paradox involved. But, the formulation of such models, as evidenced in the literature, is a very difficult task.

It is well known that the less constrained a model is, the greater its potential for reaching an optimum. The models that are formulated in the mind gain their power and their limitations from language. The power, is derived from the flexibility of language as a medium for processing and expressing relations; the limitation is derived from the slowness of language to keep pace with new situations and to discriminate between certain categories. In particular, epistemological systems refer to objects that can be described only vaguely by an essentially formal language. The situation is worsened because the approximated object-language, the model-language, and the meta-language are all the same ambiguous, natural language. As language develops, the object-language toward more richness, and the model-language toward less ambiguity, the task of building this kind of model is likely to be facilitated.

FOOTNOTES AND REFERENCES FOR SECTION 1

1. Compare C. West Churchman, "On Whole Systems," Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley, 1965.

2. Panel discussion at the "21st National Conference and Exhibit," Association for Computing Machinery, Los Angeles, California, August 31, 1966.

3. Philip M. Morse, "Queues, Inventories, and Maintenance," John Wiley and Sons, New York, 1958.

4. C. West Churchman, "Design of Inquiry," Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley, 1965.

5. See P. Armer, "Attitudes Toward Intelligent Machines," and A. M. Turing, "Computing Machinery and Intelligence." Both articles reprinted in "Computers and Thought," edited by E. A. Feigenbaum and J. Feldman. McGraw-Hill, New York, 1963.

6. If warranted this assumption may be relaxed, in which case the two levels of language could be replaced by one level.

7. K. J. W. Craik, "The Nature of Explanation," Cambridge University Press, Cambridge, 1943.

8. As shown by von Neuman in, "The General and Logical Theory of Automata," in Jeffress, L. A. (ed.), "Cerebral Mechanisms in Behavior: The Hixon Symposium," John Wiley, New York, 1951.

BLANK PAGE

2. On Models of Inquiry

In the present section a search for a theory, or at least a beginning of a theory, of mechanized inquiry is initiated. Our search effort will have to touch upon such diverse areas of science as philosophy, psychology, computer sciences, artificial intelligence, logic, etc.

Direction for the search effort is provided by the history of modern epistemology, which aids in organizing ideas and machines for inquiry into convenient classes; in addition, as Singer[1] points out, the history of epistemology also may provide direction by indicating where different proposed systems of inquiry have fallen short of their goals.

To facilitate comparison between differing types of inquiring systems, a classification of knowledge according to Spinoza[2] may prove useful. He recognized four kinds (or four levels in Polya's interpretation[3]) of knowledge:

1. knowledge arising from hearsay - mechanical knowledge;
2. knowledge arising from mere experience - inductive knowledge;
3. knowledge arising from demonstration - rational knowledge; and
4. knowledge arising from conviction that the inquirer knows - intuitive knowledge.

Spinoza and Polya both considered level 4 to be the most important. In the present context, however, the pragmatic definition of knowledge implies that the level of knowledge is of no consequence as long as potential for correct action prevails; only in cases where such action excludes certain kinds of knowledge will the categorization be of any value e.g., knowledge by hearsay will not always suffice for taking correct action in a changing environment.

Before proceeding to the discussion of epistemological models of inquiry a brief note on the language that will be employed should be given. Systems of inquiry are discussed most conveniently in teleological terms[4] and since the main source[5] for the present discussion is presented in this form, it will continue to be used. However, mechanical design of systems is likely to require a more descriptive language and therefore program or model descriptions are given in a language approximating only the teleological functional definitions.

2.1 Rationalist Inquiring Systems

The law-regulated world image of the rationalist philosophers[6] seems suited particularly for an investigation of the feasibility of mechanized inquiry. Singer's brief summary[7] provides a background of the basic ideas of the rationalists.

> To sum up, the Rationalist's argument runs in this wise: No contradiction appears in denying the hardest fact known to us by observation, such knowledge may always be doubted. And no less open to doubt must be any empirical rules generalized from such observed facts. Resting their appearances of universality on induction, these rules can obviously be no more hard-and-fast than are the facts on which they depend. But laws, necessary truths' as the school called them, are as inexorable and undeniable as the principles of logic by which they are established. By which alone they are established, - for it comes to that. As independent principles, the axioms of the special sciences will have disappeared. Only logic remains as the modus vivendi - the unique method of attaining to truth.

A system of inquiry based on these ideas was described by Leibnitz in his Monadology. However, our purpose is not to describe particular historical systems but rather to investigate their major contributions to epistemological theory. A brief account of a generalized Leibnitzian inquiring system by Churchman[8] will illustrate rationalist ideas.

By reference to the generalized Leibnitzian inquiring system hopefully two important questions will be answered partially.

a) Which level of knowledge can be created by a rationalist inquiring-system?

b) To what extent can a rationalist-inquiring-system be implemented as a program for a digital computer?

An answer to the first question will be deduced from the properties of the system described by Churchman. The second question will be answered by reference to work performed within artificial intelligence and mathematical programming.

Before proceeding to a description of the proposed system an important contribution by Leibnitz should be acknowledged. He was one of the first philosophers[9] to realize the need for a universal language of logic. Also he designed such a language to relieve his logical processor from the difficulty of resolving ambiguities of natural language.

A Leibnitzian Inquirer

Churchman's generalized Leibnitzian inquiring system is presented and commented from the aspect of mechanical inquiry. The following functions are required in a Leibnitzian inquiring system.

1. "An internal guarantee that generated results will converge."

This requirement is derived from Leibnitz' insistence on innate ideas. For the practical design a necessary condition must be met—the domain of inquiry must be described in a decidable formal system.

2. "A capability of producing strings of symbols that can be broken down into recognizable units."

The input sector can notice, individuate, break down into units, and dispatch strings of sentences for logical processing. Strings of sentences are introduced to the system (as perceptions)[10] or generated within the system. The importance of the design of the input sector was realized clearly by Leibnitz, this will be discussed further in a later section.

3. "A capability of establishing falsity or truth of any unit."

A logical processor and a dictionary of definitions are employed to establish truth (tautology) or falsity (self-contradiction) of received units. If these can't be established directly, the unit is a 'candidate' for further processing.

4. "A capability of forming nets of units by means of a given set of relations and operators."

The main deductive structure of the Leibnitzian inquirer is provided by a memory, organized for direct and chained addressing. From the beginning the memory is blank, but gradually it will build up nets of related units. For each received 'candidate unit' the memory is searched for logically related units. If the search is successful, the related units are connected and the 'candidate unit' becomes a contingent truth. Now, truth for a Leibnitzian inquirer is an end-result, which means that all sentences are doubtful until their connection with the largest fact-net is established. This makes the problem of inefficient generation of new connections acute. A procedure for directing such generation is needed. At this point, it should be pointed out that there is an analogy between the procedures of Leibnitzian inquirers and the representation of problem solving as a tree searching.[11]

The relation nets of the inquirer correspond to tree structures that are built without concern for di.ection, i.e., they are not generated selectively. It is, however, known that by exhaustion of all combinatorial possibilities, will one net eventually provide a path between the "true" definitions and the "desired" end result. The situation strongly resembles that described by Newell, Shaw, and Simon as the "British Museum Algorithm."[12]

To provide selectivity in generation of strings of units, Churchman has added two requirements.

5. "A capability of ranking the nets according to a prescribed criterion"; and

6. "A method of processing symbols and building nets based on the ranking, such that the system will eventually arrive at an optimal net and know when it has arrived. Or else will converge to an optimal net and will know that it is converging."

Requirements 5 and 6 are satisfied by introducing an "executive" which assumes the responsibility of reducing unfruitful generation of sentences to a minimum. The executive function, as conceived here has been a major design problem for artificial intelligence machines; this problem will be reviewed briefly later. Our present concern is the question of how powerful the executive can be permitted to be within a truly Leibnitzian design.

The tasks of the executive are represented in requirements 4, 5, and 6. A brief discussion of the respective functions follows:

Requirement 4 permits the inquirer to apply alternative logical systems.[13] Some choice has to be made about which system to use.

This decision is quite complicated unless, of course, all approaches can be tried out in parallel. It must be decided if the executive should be permitted to conduct inquiry on the structure of its own processing to learn about when to apply particular logical systems, or if a priori decision rules should be applied.

Requirement 5 necessitates an evaluation procedure for created nets. Such evaluation may be made in terms of computation time, simplicity, elegance, etc. It should be observed, however, that simplicity and similar measures are not defined objectively and, as such require a detailed investigation before they can be applied.

Requirement 6 can be implemented in certain areas, such as theorem proving, where answers to well-defined questions can be tested for correctness. In general, however, this requirement is difficult to implement.

2.1.1 Leibnitzian Machines

In the following, inquiring systems, which have been implemented in the form of programs for digital computers, will be called 'machines'. Several classes of Leibnitzian machines are described in the literature, examples are computer programs for mathematical programming, optimization techniques, artificial intelligence, etc., a few of these will be discussed briefly.

A majority of all computer-programs can be considered to be algorithmic-machines—i.e., to belong to a class of machines that have been proven to find solutions to all problems within a specified domain, and to do so in a finite time. Typical examples of such machines are linear programming programs, programs for finding maxima or minima, programs to solve systems of equations, etc. The Leibnitzian requirements are fulfilled in that[14]:

1. the machines operate within a decidable formal system;
2. the machines have access to primitives and rules of sentence-formation of the formal language;
3. all definitions and rules of inference are available to the machines;
4. an implication net is formed by strings of symbols connecting given inputs with the desired result;
5. the time requirements of alternate paths of processing may be determined; and
6 there is some way of checking if a solution is obtained

Search-machines[15], such as steepest ascent or other gradient search machines, pattern-search machines, box-analysis machines, etc., satisfy the Leibnitzian requirements under the following two conditions:

1) their environment is unimodal, and
2) a solution, or an optimum, is defined as a range around the 'real' optimum.

A special class of these machines are 'heuristic' search machines such as the line-balancing program by Tonge[16] and other programs for manipulation of combinatorial situations.

A class of programs, which follow very closely the Leibnitzian description, is the "simple deduction machines", or question-answering machines, exemplified by Raphael's SIR,[17] Slagle's DEDUCOM,[18] McCarthy's Advice Taker,[19] and others. In these machines simple facts formulated in a formal language are connected to implication nets in such a way that simple questions can be deduced by tracing through relevant paths.

A typical problem solved by DEDUCOM is the "Monkey Question". The following facts are given:

1. The monkey can move the box to any place;
2. Someone moving v to u leads to v being at u;
3. The monkey can climb the box;
4. v being at u and p climbing v leads to v being at u and p being on v;
5. Under the bananas is a place;
6. If the box is under the bananas, and the monkey is on the box, then the monkey can reach the bananas; and
7. p reaching x leads to p having x.

Question:

"What should the monkey do so that the monkey has the bananas?"

DEDUCOM'S ANSWER:

```
((THE MONKEY SHOULD DO THE FOLLOWING)
(THE MONKEY MOVES THE BOX UNDER THE BANANAS)
(THE MONKEY CLIMBS THE BOX)
(THE MONKEY REACHES THE BANANAS))
```

An important group of Leibnitzian machines are the theorem-proving machines exemplified by those of Wang[20], Davis & Putnam,[21] McCarthy,[22] Robinson,[23] Friedman,[24] Gilmore,[25] Wos, Carson & Robinson,[26] and others. Such machines have been designed to employ:

a) Decision procedures, which are available for propositional calculus, elementary algebra, elementary theory of conditional expressions, and certain other branches of mathematics.[27]

For undecidable calculi, such as the first-order predicate calculus[28], the machines may employ[29]:

b) Decision procedures for solvable subclasses of undecidable calculi;

c) Proof-procedures that will recognize any theorem, but will not converge for non-theorems[30]; or

d) Semi-decision procedures, which approaches b and c are combined, i.e., provide decision procedures for solvable subclasses but proof-procedures outside there.

Although theorem-proving machines perform satisfactorily in certain domains, the problem arises of an exponentially-growing time requirement of processing for linearly-increasing numbers of clauses or connectives, limiting their applicability to rather simple theorem proving.

Complex-Search Machines, as exemplified by GROPE[31] by Flood and Leon, utilize alternate logical processors. The processors are applied to problems according to rules derived from their performance in previously encountered situations. In the case of GROPE, the executive employs a simple symmetric stochastic learning model to choose a logical processor. Progress in problem solving is measured by criteria of relative improvement in a hill-climbing situation. The machine exemplifies in this way the application of a sophisticated executive in a Leibnitzian inquirer.

Our brief review of typical Leibnitzian machines indicates clearly that much work has been done in this area. This has lead to the development of efficient methods for frequently encountered tasks. In this context only the importance of efficient tree-searching methods, such as the $\alpha\ \beta$ procedure[32] employed in several artificial-intelligence machines will be stressed.

2.1.2 Knowledge Created by Leibnitzian Inquirers.

An important contribution of the rationalists was their insistence on removing ambiguities caused by natural language and introducing instead precision in the form of a universal language of logic. However, the employment of an autonomous logical processor precluded any investigation of the nature of the processed units, thus making the results of inquiry precise but empty of content.

Given an internal guarantee of convergence, the Leibnitzian inquirer is purely deductive and can thus be classified as producing knowledge of Spinoza's third level. However, the inquirer guarantees that, given a valid input-sentence, the 'correct' answer will be produced. Therefore, for an external observer, or user, its process of producing knowledge can not be discriminated from a "dictionary-look-up", i.e., from "hearsay-knowledge". Thus, Spinoza's third level,[33] "perception arising when the essence of one thing is inferred from another thing, but not adequately" is not reached. It should, however, be noted that our pragmatic search for knowledge is not concerned so much with how it was achieved as with what it means. Therefore Leibnitzian inquiring systems, even if looked upon as huge dictionaries, are valuable systems, not only when used separately, but also when employed as logical processors in more sophisticated inquirers.

2.1.3 Representation in Leibnitzian Inquirers

The problem of representation is only encountered at a superficial level in the Leibnitzian inquirer, in fact, one to one correspondence between symbolical representations and unambiguous raw-data precludes any problem-oriented search for "optimal" representations. Only in the case of complex machines, e.g., GROPE, is an implicit choice among alternate representations made. As such selectivity is the major feature of Kantian inquiring-systems a detailed discussion will be postponed.

2.2 Empiricist Inquiring Systems

The empiricists,[34] in opposition to the emptiness of content of the rationalists' system of formal inquiry, presented a theory of inquiry based on the "reality of things." At first glance their theory appears very reasonable, in particular as it seems validated by common sense feeling as well as psychological evidence of basic learning processes.[35] The empiricists' thesis may be summarized in Locke's words:

> These two, I say, viz. external material things as the objects of sensation and the operations of our own minds within as the objects of reflection, are to me the only originals from whence all our ideas take their beginnings.[36]

The empiricists' theories of inquiry have been criticized, and deservedly, by later schools of epistemology. However, as we are not giving a critical survey but are searching for ideas, an amended version of a Lockean inquiring system will be employed to represent the major ideas of the empiricists.

From a Lockean inquiring system we require:

1. a capability of receiving and individuating inputs

The input-sector acknowledges reception of input-units (sensations) and individuates the unit by space-time-coordinates. The empiricists failed to realize the importance of this sector, Singer points out:

> "However sound may be the Empiricist's account of how our knowledge has grown once having started, his own account of this growth makes it impossible for its start to be part of it".[37]

2. a capability of labelling and transmitting received inputs

The inp.t pre-processor recognizes the unit as simple or complex and labels the unit by attaching a list of properties corresponding to impressions from sense-organs. The pre-processed unit is transferred to the memory

The task of the pre-processor is to define ideas, Churchman and Ackoff[38] write

> within empiricism progress in defining was "theoretically" possible. To make a definition better, the references to the immediate had to be made more precise. Insofar as we could make that which was designated by a word less and less ambiguous, the definition of it could be made better and better. But as defining became better and better with respect to "content", for the empiricist, it became worse and worse with respect to communicability, since content was a function of the "immediate" which was itself inexpressible

In requirement 2 the empiricist manner of defining becomes a labelling operation performed by a filing-system which can grow its own categories Such categorization is performed by an artificial intelligence machine, EPAM[39], which will be described in section 2.2.1.

3. a capability to reflect on the internal processing.

One of the functions of the executive is to observe and label the internal processing of the inquirer.

4. a capability to perform logical operations on labels.

The "acts of mind" according to Locke are combining, comparing, and abstracting. Thus the logical processor incorporates facilities for compounding labels by logical connectives. A logic of classes is employed.

5. a capability to generalize on experienced sensations.

The generalization sector is globally applicable. It complements the input preprocessor in making abstractions necessary for establishing similarities of inputs. The logical processor as well as the executive are possible domains for generalizations.

6. a capability to communicate about labels.

The Lockean inquiring-system needs a guarantor of reality to replace the innate ideas of Leibnitz. Locke stated;

> Our knowledge, therefore, is real only so far as there is a conformity between our ideas and the reality of things[40]

Such conformity according to Locke was present in his design. Simple ideas were imposed upon the inquiring-system by reality and since complex ideas were internally created for internal processing, they had no reason to be connected to reality. Complex inputs (substances) caused complications, but an involved reference to their parts established conformity.[41]

Although the empiricist inquiring-system may have a tie to reality via the naming of simple sensations, there is no guarantee of validity unless a community of inquirers can agree on the correct label of the input. Locke seems to feel that the labels of different inquirers are isomorphic, because he writes: "For what need of a sign, when the thing signified is present and in view."[36] Churchman,[42] however, points out that it is a non-trivial task for a community of inquirers to agree that they are talking about the same input when assigning its name. Community consensus thus is employed as a method of ostensible definition of meaning.

2.2.1 Design of Mechanical Lockean Inquirers

There are three essential problems to be met in the design of Lockean inquiring systems, namely

1) to define simplicity,

2) to represent generalization,

and 3) to design a non-trivial guarantor-function.[43]

Simplicity of sensations may seem to be easily defined, in particular, when there are words available to define simple concepts. However, as simplicity is a property of particular situations, experience may require further breakdown of concepts which initially were considered simple. Thus the Lockean inquirer must have a capacity for changing its concepts of simplicity. Such capacities are easily implemented in internal processing languages, but are very difficult to handle in the communication language. EPAM, a computer model of human learning by E. Feigenbaum[44] is an example of a Lockean filing system which creates its own categories.

> The EPAM program is the precise statement of an information processing theory of verbal learning that provides an alternative to other verbal learning theories which have been proposed.[45]

In the present context we are less interested in EPAM as a model of verbal learning than in its capacity for efficient categorizing, storing, and retrieval of information. The primary information structure in EPAM is the discrimination net, a sorting tree which is grown to accommodate all information encountered in the problem-solving process.

An example of a discrimination net is given in Figure 2.1.

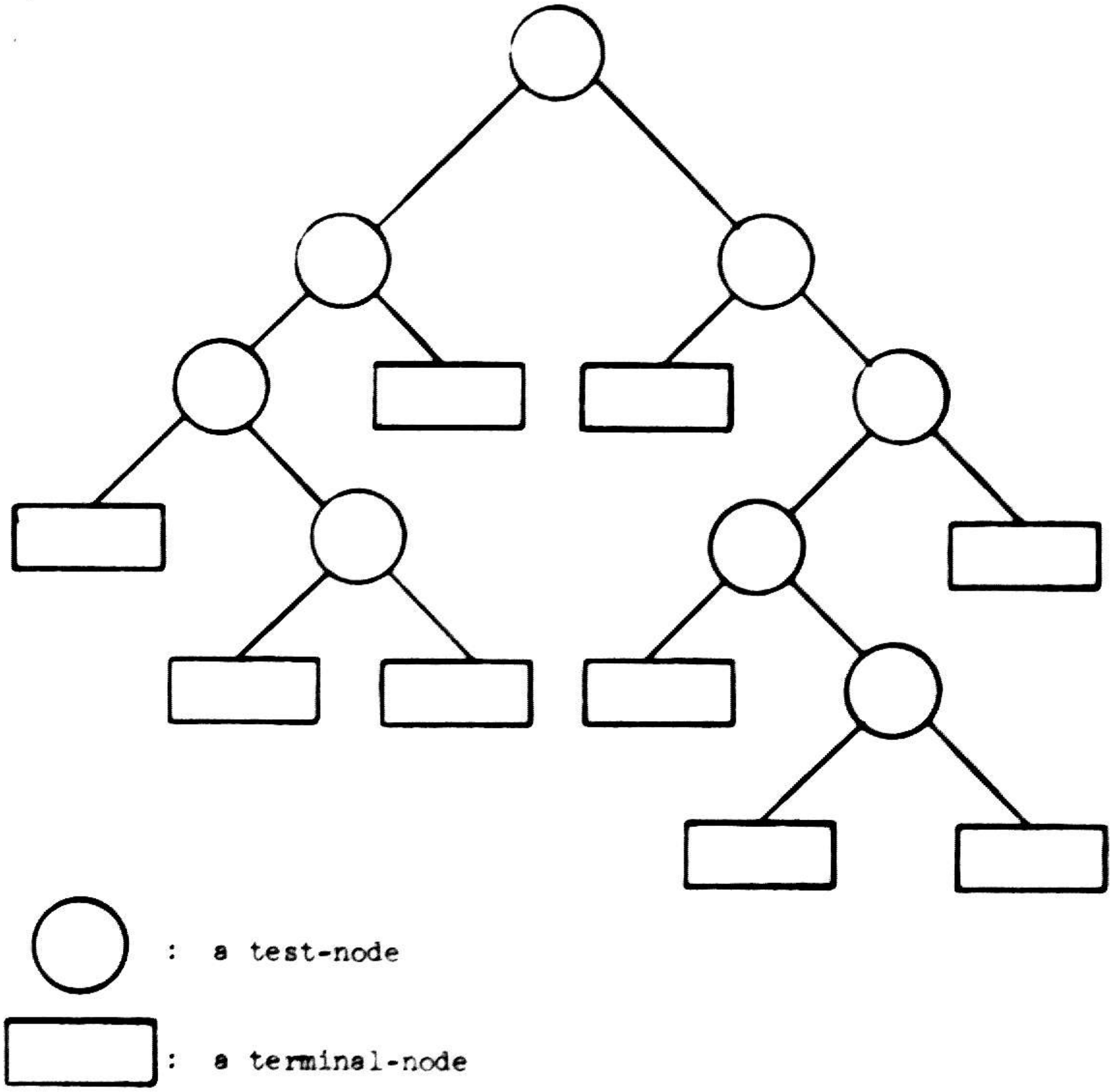


Figure 2.1

Information is stored at the terminals of the net and tests are placed at the nodes. Labels and/or property lists associated with received entities are tested at the nodes and the entities, as a result, sorted down left or right.

For retrieval of stored information only those properties of the label which have been tested during the procedure have to be presented to the net.[46] If an attempt to store information at a previously occupied terminal is made, a further discrimination has to be initiated, and a new test is added, (Figure 2.2)

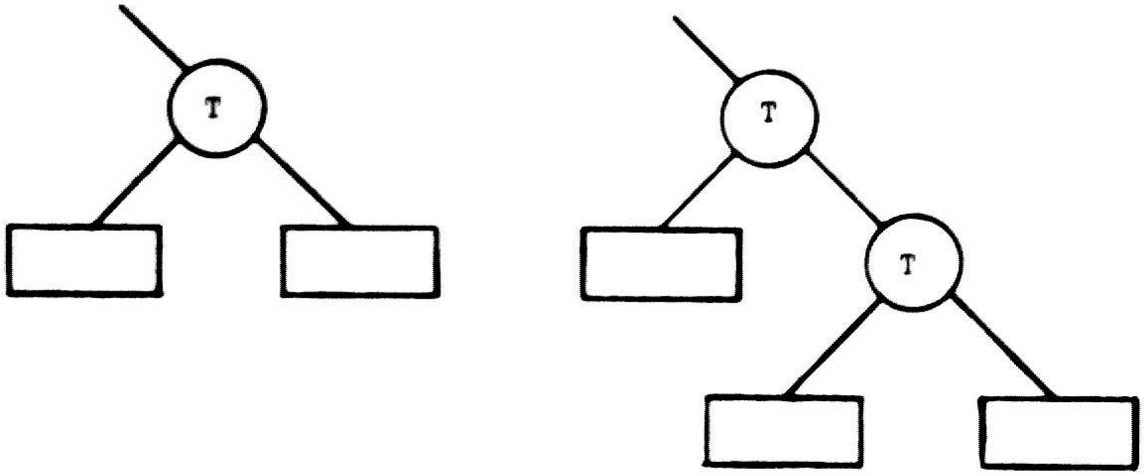


Figure 2.2

A simple example: Suppose we have translated derived property lists to binary labels which are introduced to the discrimination net, then it may grow as exemplified in Figure 2.3.

Ti = i:th digit test. (i = 1,2,3,4) .

1. Store 0100

2 Store 1011

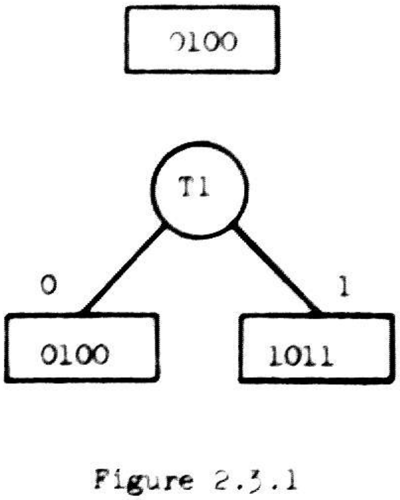
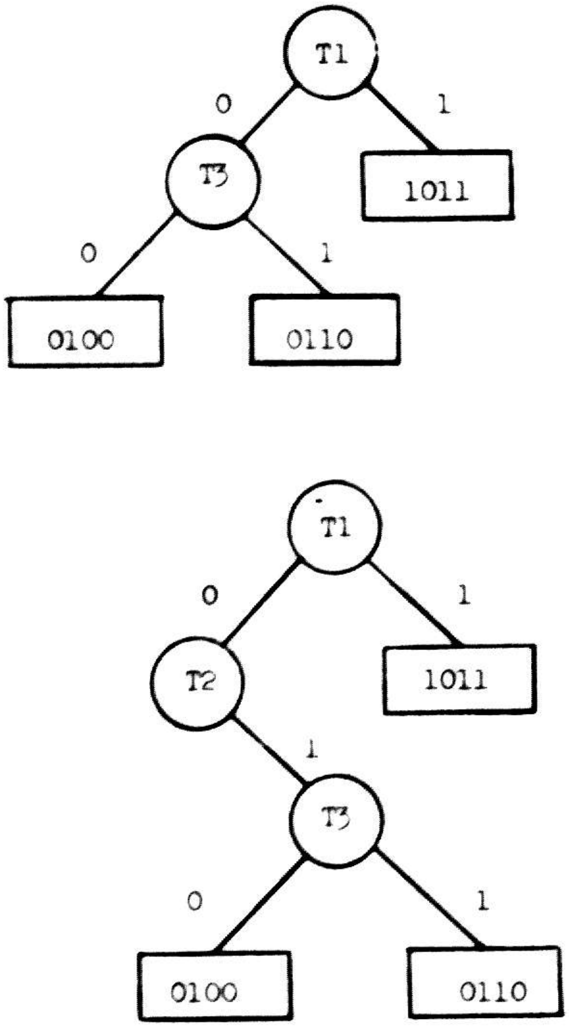


Figure 2.3.1

3. **Store** 0110



or

Figure 2.3.1

Store 0010

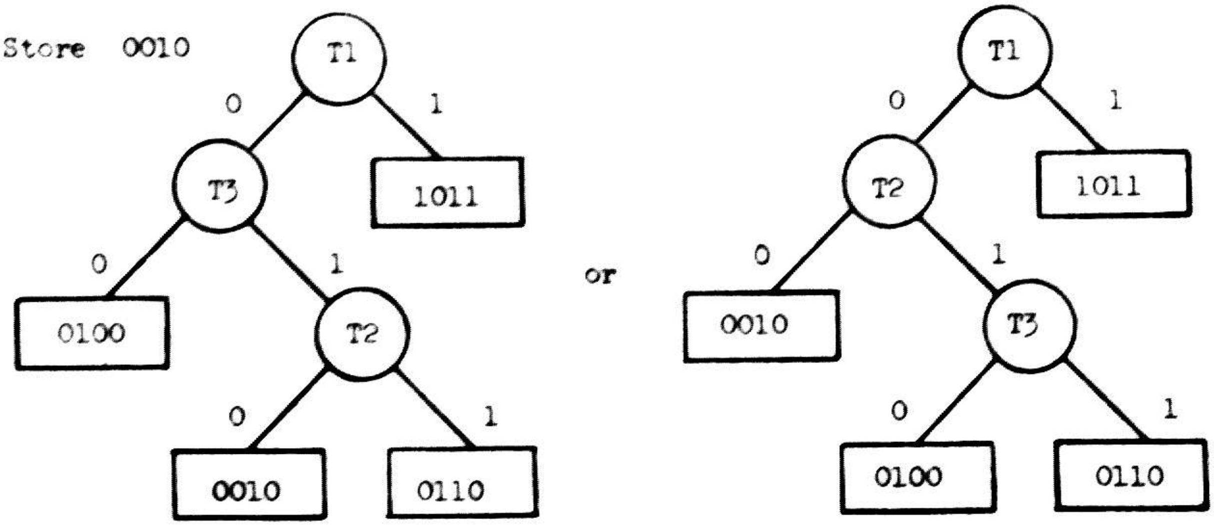


Figure 2.3.2

EPAM clearly will never make a finer categorization than is necessary at the moment of receiving an entity. This means high efficiency of tree-searching and low requirements of storage. However, an amended EPAM would be desirable for the second problem, i.e., description of generalization.

Generalization in a Lockean inquirer means grouping of concepts into classes. Such grouping can be done synthetically by adding concepts into generic classes, or it can be done analytically by considering incomplete breakdowns of complex concepts. The situation is most conveniently represented in a tree, where the terminal nodes are simple concepts and all other nodes define sub-trees which are classes of concepts. Thus at the trunk of the tree there is a class 'something', which by tracing the tree will be broken down into successively finer classes. In the case of EPAM, the terminal nodes represent desired groupings provided that a proper design and noticing-order of tests is implemented. In Figure 2.7, there are two possible classifications containing 0110, namely {0100, 0110} and {0010, 0110} . The correct grouping can only be determined by reference to a particular context, but since the possibilities for ambiguities to arise must be kept in mind, a carefully planned noticing-order is maintained.

In mechanized inquiry the important guarantor-function "community-agreement" will have to be replaced by some "conventional"[42] method of validating that received inputs are representations of reality. One such "conventional" design is to embed the inquirer in a larger system which guarantees the validity of its communications to the inquirer. A non-trivial requirement would be that the proposed system has to agree with its own labeling over time. This requires a capability to abstract from changes which are functions of time, and, therefore, puts pressure on the capability of individuating inputs.

2.2.2 Knowledge Created by Lockean Inquirers

The level of knowledge achieved by a Lockean inquirer depends upon the nature of its guarantor-function. If the guarantor is "conventional," then the Lockean inquirer does not go any further than the Leibnitzian, because the "sensations" of the former do not correspond to reality any more than the perceptions of the latter.

2.2.3 Representation in Lockean Inquirers

The major weakness of the empiricist inquiring system is to be found in the input-sector. Although a claim of correspondence between labels and "reality" is made, physiological and psychological evidence shows that the "sensations" of reality by no means are unambiguous, in particular, the individuation of objects, the intensity of impressions on the sense organs, and the range of the sense-impressions are not uniquely defined; furthermore, there is no representational function to guide the observations on the real world. Thus the representational functions of Lockean inquirers are not sufficiently sophisticated to support directed search for information.

2.3 The A Priori Sciences of Inquiring-Systems

An attempt to bring the rationalist and empiricist systems of inquiry together was made by Kant and other criticists. We have given rather detailed descriptions of Leibnitzian and Lockean inquiring systems. These systems employ most of the components of the Kantian inquirer. One major sector, however, remains to be discussed, namely, the role of a priori sciences.

Kant's main contribution to epistemology may well be that he realized that "whenever there is experience, there are prerequisites of experience." This realization led him to conclude that the sciences necessary for recognition (geometry) and individuation (logic) of phenomena had to be a priori to any experience. Post-Kantian developments have introduced the possibility of introducing alternate a priori-sciences, i.e., to employ some geometry and some logic. The main difficulty of the Kantian doctrine, however, still remains: to establish how the a priori enters the mind. There have been several attempts to form such a theory ranging from Plato's deduction of a previous existence to theories of psychological a prioris. However, none of these, including Kant's has been convincing. In the present pragmatic search for ideas, however, this is of no concern, because, if the a priori can be shown to be a necessary attribute of inquiring-systems, then feasible methods of their implementation, not their actual origin, is of importance.

The availability of alternatives for the choice of the a priori suggests the design of a Kantian inquiring-system with multiple sets of a priori sciences. As Kant did not imply that the a priori sciences were simple, the possibility of employing powerful a priori comes forth as a viable alternative to previous parsimonious attempts. Churchman, in his discussion of what he calls maximal a priori's, states:

> Suppose, instead, we were to say that there is no "basic" mode of representation in the design of the input sector of the inquirer, and that a maximum flexibility in representation is desirable. Hence instead of attempting to minimize the influence of the a priori of the inquiring system on the information, the designer should try to maximize this influence in order to represent the information in a manner in which the problem solution becomes facilitated.[47]

Parsimony in a priori sciences, however, permits unbiased generality but is very likely to lead to inefficient problem-solving. Therefore, a judicious balance between domain and efficiency of inquiry will have to be maintained in order to permit an "optimal" degree of specialization.

The tasks of the executive of a Kantian maximum a priori inquiring-system will have to be more sophisticated than in systems previously discussed. In particular, the availability of alternate a priori sciences will require decisions to be made on how to represent particular problems. Thus the task of the Kantian executive is to translate a problem-formulation into the language of a suitable model, i e., relate its object-domain to a model. Other tasks of the executive are to select input for processing and to judge the relative difficulty of differently represented problems. These tasks will be discussed in depth in Sections 3, 4, and 5 Instead of anticipating this discussion let us now turn to a more sophisticated system of inquiry based on Singer's ideas.[48]

Singer was dissatisfied with the "paradox of a priori" and, by observing the methodology of empirical and formal sciences, came up with the system of inquiry presented in his "Experience and Reflection". Singer's "Experimentalist System" is based on a careful distinction between what is known to experience and what is known to reflection on experience. Experience belongs to one subject, the learner, but the a priori possessions to another, the reflective mind. Thus, an inquirer can be said to produce knowledge only if it is observed doing so by another inquirer, and the designer of an inquiring-system is a member of the very same system, because his prepossessions are reflected in its design. The position of the experimentalist is to relate the formal and the non-formal by their common purpose, progress of science. Churchman and Ackoff[49] summarize the experimentalist ideas as:

> The precision with which we can respond to a question is a function of the precision with which we can ask a question. The latter is itself a function of the explicitness of presuppositions of the asking and answering of a question. The actual number of presuppositions involved in framing any question is indefinitely large. It has been the lesson of experimentalism that the final answer of any question presupposes the final answering of every other question. The absolute answer to a question is an ideal which may be constantly approached but never attained. Consequently, the presuppositions, as well as the response, are constantly subject to change. Science is capable of progressive change insofar as it can indefinitely reduce the error expressed in the response to any question.

Thus Singer's experimentalist inquiring-system employs a technique of experimental control that will make a progression of answer-question pairs to converge to a limit - an a priori fact. Although the Singerian system of inquiry is extremely hard to visualize in a non-trivial mechanized system, a slightly Singerian flavor is added to our sequence-extrapolating system by permitting its executive to inquire about the problem-concocter in order, hopefully, to be able to predict his behavior.

2.3.1 Representation, a function of the a priori

The Kantian inquiring-system stresses the importance of a priori sciences, which in turn stresses the importance of finding suitable representations of problems, because, the presuppositions, i.e., the a priori sciences, determine what information to ask for, i.e., the representation of raw data. The representational functions of the Kantian inquirer will be discussed in Section 3. A more specific context, namely sequence extrapolation, is the subject of Sections 4 and 5.

2.4 Summary

A few epistemological models have been reviewed. Although several alternate theories could have been presented, our choice has been based on representativity of schools of thought as well as on potential for computer-implementation. Thus an interesting inquirer - the Hegelian - has not been reviewed, because the dialectic method at present seems too much of a speculative tool for programming on a computer.

The systems reviewed and especially, the Kantian, have contributed to our understanding of efficient organization and classification of mechanized inquirers. The later developments, including Singer's system provide a direction for search of feasible ways of implementing more sophisticated inquirers. In the next section, the problem of representation raised by the Kantian maximum *a priori* approach will be discussed.

FOOTNOTES AND REFERENCES FOR SECTION 2

1. E. A. Singer, Jr., "Experience and Reflection," 1959, University of Pennsylvania Press.

2. B. de Spinoza, "On the Improvemement of Human Understanding," The Chief Works of Benedict de Spinoza, Vol. II., tr. Eleoca, R. H. M George Bell and Sons, 1884, reprinted in Dover edition, 1955.

3. G. Polya, "Mathematical Discovery," Vol. I., John Wiley and Sons, Inc. New York, 1962.

4. See M. Minsky, "Steps Toward Artificial Intelligence," Proceedings of the Inst. of Radio Engineers, Jan., 1961, 49:8-30.

5. C. West Churchman, "Design and Inquiry," Internal Working Paper No. 28, Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley.

6. Here represented by Descartes, Leibnitz, and Spinoza.

7. Ref. [1], p. 16.

8. C. West Churchman, "Rationalist Inquiring Systems," Internal Working Paper No. 29, Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley, 1965.

9. Others are exemplified by Russell and Carnap.

10. Leibnitz Monads generated all perception-streams from within.

11. See any of a number of papers in Feigenbaum and Feldman, ed., "Computers and Thought," 1963, McGraw-Hill.

12. A. Newell, J. C. Shaw, and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics." Proc. of the 1957 Western Joint Computer Conf., Feb., 1957, p. 218-230, reprinted in ref. [11].

13. Such as propositional calculus, predicate calculus, principia mathematica, modal logic, etc.

14. See ref. [8].

15. See D. J. Wilde, "Optimum Seeking Methods," Englewood Cliffs, N. J., Prentice Hall, 1964.

16. F. M. Tonge, "A Heuristic Program for Assembly Line-Balancing," Englewood Cliffs, N. J., Prentice Hall, 1961.

17. B. Raphael, "A Computer Program Which 'Understands,'" Proc. Fall Joint Computer Conf., 1964, Vol 26, Spartan Books.

18. J. R. Slagle, "Experiments With a Deductive Question-Answering Program," Communications of the ACM, Vol. 8, No. 12, Dec., 1965.

19. T. McCarthy, "Programs With Common Sense," Symp. Mech. of Thought Processes, Nat. Phys. Lab., Teddington, Middlesex, England, 1958.

20. H. Wan, "Towards Mechanical Mathematics," IBM Journal of Research Dev., Vol. 4, 1960, p. 2-22.

21. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," Journal of the ACM, Vol. 7, 1960.

22. J. McCarthy, "Computer Programs for Checking Mathematical Proofs in Recursive Function Theory," Proc. of Symposia in Pure Mathematics, Vol. 5, Amer. Math. Society, 1962.

23. J. A. Robinson, "A Machine Oriented Logic Based on the Resolution Principle," Journal of the ACM, Vol. 1, Jan., 1965.

24. J. Friedman, "A Computer Program for a Solvable Case of the Decision Problem," Journal of the ACM, Vol. 10, No. 3, July, 1963.

25. P. C. Gilmore, "A Proof Method for Quantification Theory," IBM Journal of Research Dev., No. 4, 1960.

26. L. Wos, D. Carson, and G. Robinson, "The Unit Preference Strategy in Theorem Proving," Proceedings of the Fall Joint Computer Conf., Vol. 26, Spartan Books, 1964.

27. See ref. [22].

28. Proven to be undecidable by Church, 1936.

29. See ref. [24].

30. For undecidable calculi constraints on the time requirement for finding a proof, at least terminates the process of generation of sentences.

31. A. Leon, "Steps Toward a Universal Adaptive Code for Optimization," (GROPE), Internal Working Paper No. 11, Space Sciences Laboratory, University of California, Berkeley.

32. T. Hart and D. J. Edwards, "The Tree Prune Algorithm," M.I.T. Artificial Intelligence Project Memo 30, Dec., 1961, M.I.T. Comp. Center.

33. See ref. [2].

34. Here, mainly represented by Berkeley, Hume, and Locke.

35. Compare EPAM (ref., see note 44) as a theory of verbal learning.

36. J. Locke, "An Essay Concerning Human Understanding," Book II, Chap. 1:4, Dover Publ, New York, 1959.

37. See ref. [1].

38. C. West Churchman and R. L. Ackoff, "Psychologistics," University of Pennsylvania, 1947, mimeo.

39. E. A. Feigenbaum, "The Simulation of Verbal Learning Behavior," in "Computers and Thought," edited by E. A. Feigenbaum and J. Feldman, McGraw-Hill, New York, 1963.

40. Ref. [36], Book 4, Chap. 4:3.

41. The connection between ideas and reality was stressed by the empiricists. In particular, Hume and Mill stressed the reduction of complex ideas, or words, to simple.

42. C. West Churchman, "Lockean Inquiring Systems," Internal Working Paper No. 45, Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley, 1966.

43. See ref. [42].

44. See ref. [39].

45. Unless the growth of the discrimination-net necessitates extra information. Compare: E. Feigenbaum and H. A. Simon, "Forgetting in an Associative Memory," Proc. of the ACM Nat. Conf., 1961.

46. See ref. [42].

47. C. West Churchman, "Kantian Inquiring Systems," Internal Working Paper No. 46, Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley, 1966.

48. See ref. [1].

49. See ref. [38].

3. On Representation

Inquiry and problem-solving generally raise the following questions:[1] How are the premises collected? How does deduction proceed once they are found? These questions are strongly interdependent in view of the problem of representation even though, according to the positivists and statisticians, they could be attacked and solved separately. A third question, therefore, will have to be posed, namely, how to represent the problem.

The subject of representation reaches far It connects the *a priori* theory necessary for observing the environment with the logical processing necessary for coming to any conclusions about it. Our present task, however, will restrict the discussion of representation to a technical level by exploring it as a prerequisite for mechanized inquiry.

We have not asked if symbolic representation is really necessary at all For machines and for most human inquiries, it is. Even though Henri Bergson[2] describes an intuitive knowledge which arises without employing symbolic representation, we have no reason to discount its importance for inquiry.

The evidence in favor of representing reality by symbolism is too massive to permit us to doubt its importance Evidence is available from different branches of science, and we will briefly indicate some sources. Susan Langer states, "A new philosophical theme has been set forth to a coming age. An epistemological theme, the comprehension of science The power of symbolism is its cue, as the finality of sense-data was the cue of a former epoch."[3] She reviews in detail the literature on philosophy of language, which clearly indicates that the subject of representation is important in contemporary philosophy. In psychology, we need only to refer to works such as Bruner, Goodnow, and Austins: "A Study of Thinking,"[4] or Humphrey's "Thinking"[5] to establish the relevance of language. Even neurology, anthropology, linguistics and other branches of science, support the conclusion of A. D. Ritchie that, "as far as thought is concerned, and at all levels of thought, it [mental life] is a symbolic process."[6]

The most elementary form of symbolism is the action, or push-button, response to signals, which corresponds to Spinoza s classification "hearsay-knowledge." This is in animal behavior represented by "built-in subroutines"[7] and conditioned reflexes The level of knowledge sought in sophisticated inquiring systems, however, requires far richer representations than these pure signal-action relations permit,[8] thus epistemological models, as previously shown, require elaborate languages for internal processing as well as for communication

Granted that elaborate representations are required, then the question of which one to use is relevant unless, as is the case between the languages of Western culture, translation can be performed without actually distorting very much information[9] Such translation is, however, not possible between languages of very different cultures, for example the detailed categorization of snow in the language of the Laps differs in degree and the space-time relations in the language of the Hopi-Indians, differ in kind from corresponding parts of the English language. The linguist Whorf from these facts drew his hypotheses that,

> We are thus introduced to a new principle of reality which holds that all observers are not led by the same physical evidence to the same picture of the universe, unless their linguistic backgrounds are similar or can in some way be calibrated.[10,11]

The observation of Kant that people put regularity into nature is obviously supported by the fact that people actually see reality differently, depending upon their language. Now if the world-image is a function of its representation in terms of language we clearly should choose the "best" possible representation. Unfortunately, as there is no objective measure of the goodness of language, comparisons of efficiency can be made only along a few dimensions. Thus a language of internal processing should be unambiguous as Leibnitz, later Russell, and in particular Carnap, in his "Logical Syntax," have observed. For other dimensions the vague notions of simplicity, efficiency, etc., will have to be employed.

Language obviously provides a model of our world image and helps us to observe and inquire into reality. However, language can be used to design alternate models of specific domains of inquiry and therefore the problem of choosing an "optimal" representation arises. Churchman has described the situation in his discussion of the maximum a priori approach to the design of Kantian inquiring systems.[15]

Within our language are several sub-languages relevant to description of particular domains, so that events or problems are presented in a form accessible for "logical processing." Such models, or grammars, are abundantly represented in the progress of diverse sciences. Several branches of science use isomorphic models, which has led to speculations about universality of scientific laws or similarities between sciences as suggested by Von Bertalanffy,[12] in "An Outline of General Systems Theory" or, as expressed by McKay, in his statement "Many scientific concepts in different fields have a logically equivalent structure.

One can abstract from them a logical form which is quite general and takes on different peculiar meanings according to the context."[13]

If such universality exists, then of course one a priori would suffice, but specialized models are still likely to be preferred for reasons of efficiency and/or simplicity. The importance of scientific representation cannot be overemphasized. We cannot here give a comprehensive discussion but will cite a few examples:

1. Kinematics by means of Newton s law received a new and very powerful predictive model. Although the model has been found incomplete, it still provides a useful representation on the terrestrial scale.

2. Organic Chemistry developed more rapidly after the representation of cyclical molecular structures was invented.

3. The theory of ideal gases has led to many significant inventions since the concept of entropy was introduced.

4. Astronomy has experienced significant improvements in its predictive capabilities as the result of Kepler s and Einsteins' representations

5 The important role o mathematical notation can hardly be doubted.

In summary, development of adequate representations plays a very important part in scientific inquiry, and is also reflected in the applied sciences

All models require a certain structuring of their domain of application Mulligan[14] gives an amusing account of how people with varying backgrounds try to squeeze an industrial engineering optimization problem into their particular model. One can question what 'reality is ke when the same situation can be described as a linear programming model, a Lagrangian multiplier application, a dynamic programming case, a simulation situation, a task for gradient search-methods, etc. The answer obviously must be that "realtiy" is what our thoughts makes it, and that it therefore depends upon representation.

It is obvious that, following Kant, Hegel, Singer and others, there is no 'right' in the sense of an optimum representation of a situation. Every representation is the result of some a priori theory. It carries some experience-dependent approximations etc., and we will therefore never escape the doubt of Hume about the validity of scientific reasoning or, notably, induction based on such representations.

For a pragmatic use of representation, however, the situation is different. An efficient representation is not only a simple way of handling data as opposed to an inefficient and complex way.[15] An efficient representation often means the difference between success and failure. If there has been at least one inductive step in the development of a suitable representation, then there is no guarantee that some other representation based on a minimum a priori, will ever get to the solution unless a particular history should be repeated. Therefore, mechanical inquiry will have to be based on suitable representation, a subject to which we will now turn.

3.1 Representation in Mechanized Inquiry

The present subject is most conveniently discussed in the context of particular examples. Polya[16] presents an anecdote about little Carl Friedrich Gauss still attending primary school. "One day the teacher gave a stiff task: to add up the numbers 1, 2, 3, and so on to 20. The teacher expected to have some time for himself while the boys were busy doing that long sum. Therefore, he was disagreeably surprised as the little Gauss stepped forward when the others had scarcely started working, put his slate on the teacher's desk, and said, 'Here it is!'" Little Gauss had obviously found an efficient representation of the problem such as

$$\begin{array}{ccccccccccccccccccccc}
 & 1 & + & 2 & + & 3 & + & 4 & + & 5 & + & 6 & + & 7 & + & 8 & + & 9 & + & 10 \\
+ & 20 & + & 19 & + & 18 & + & 17 & + & 16 & + & 15 & + & 14 & + & 13 & + & 12 & + & 11 \\
\hline
 & 21 & + & 21 & + & 21 & + & 21 & + & 21 & + & 21 & + & 21 & + & 21 & + & 21 & + & 21 \\
= & 10 & \times & 21 & = & 210 & & & & & & & & & & & & & &
\end{array}$$

How Gauss did it is not important, but the fact that there is a representation which simplifies the problem is sufficient evidence for us to study the problem of how to find it.

At a different level of representation problem-solving is often described as a process of tree-searching In particular, almost all currently implemented artificial-intelligence-machines[17] employ some form of tree-searching and/or tree-growing process. In general, a problem is represented as a state S_o, which by application of any of a finite number of operators O_i $(i = 1, 2, \ldots, r)$ can be transformed into other states S_j $(j = 1, 2, 3, \ldots)$ which, in turn, can be transformed to new states by applying operators, etc. A solution to the problem is defined as one particular state S_s or any member of a set of states S .

All states except S_o can be conceived as generated by some sequence of operators, operating on S_o . The totality of such sequences can be summarized in the form of a tree. Suppose there are only two operators 0_1 and 0_2 available, then the situation of figure 3.1 arises.

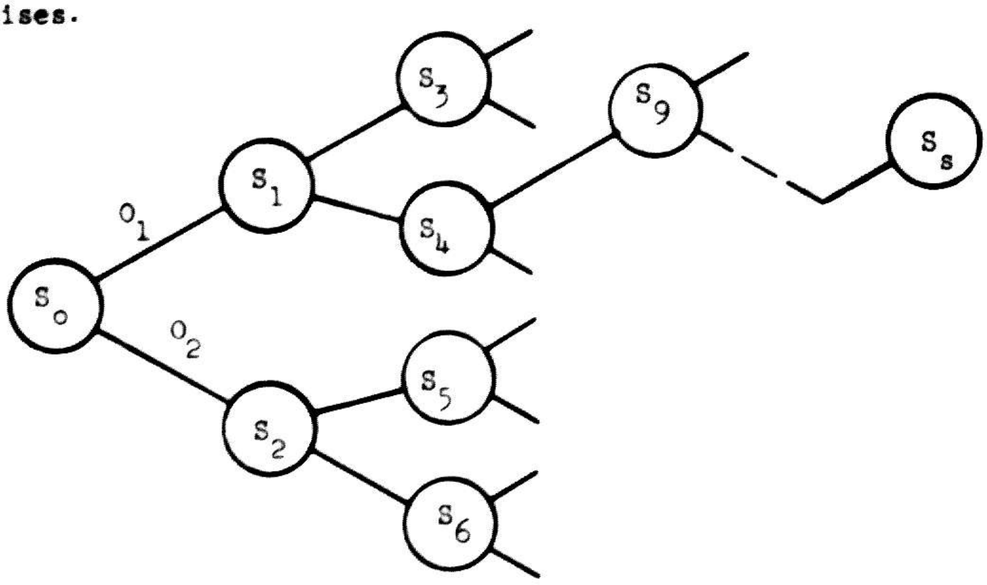
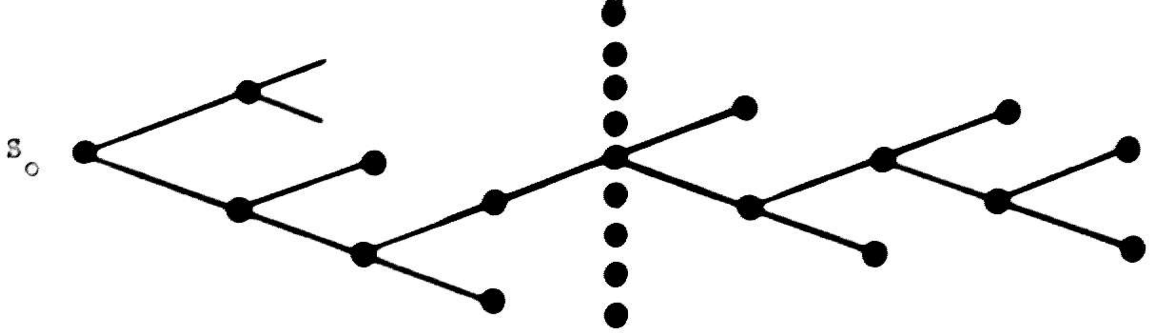


Figure 3.1

As we see, the representation of the problem S_o can be transformed to a solution S_s by a seq'ience of applications of operators.

The number of available operators may be quite large. For our purpose the tree is most conveniently assumed to be built from two compounds of operators, one context-dependent and one context-independent.



context-dependent representations context-independent

Figure 3.2

A problem thus is solved in two stages:

1. Find a solution to the context-dependent part, i.e., find a feasible representation.

2. Starting with an acceptable representation, deduce the solution of the problem.

Granted that such division can be made, Leibnitz's idea of a general logical processor can be made operational, such processor could probably be designed to perform quite well in the first order predicate calculus and for other calculi. The matter of incomputability does not necessarily prohibit the use of 'undecidable' systems, because an operational definition of failure in the form of a time- or operation-cycle-constraint makes the system 'pragmatically' decidable. Several logical calculi can, of course, be employed such as suggested in the generalized Leibnitzian system. Lindsay[18] writes, "... an intelligent machine would achieve some economy by employing general purpose representations whenever usable rather than devising special schemes for each case." This statement in the description of Figure 3.2 translates to: "Try to minimize the utilization of context-dependent operators," which amounts to pushing the line A in Figure 3.2 as close to S_o as possible. This approach is taken in several artificial intelligence machines of minimum *a priori* type, where all processing is performed in a single formal language.

Such machines are abundantly represented in the literature on artificial-intelligence. Actually, there is a whole sub-branch of artificial intelligence, which concerns itself with the design of such processors. Some examples are theorem-proving-machines, (see Section 2.1.1), "Simple deducers," such as Slagle's DEDUCOM,[19] Newell, Shaw and Simon's

LOGICAL THEORIST,[20] and the ADVICE TAKER by McCarthy;[21] and so-called question answering machines such as Raphael's SIR,[22] and Bobrow's[23] STUDENT, which compile English-like sentences into the language of the logical processor.[24]

The converse of the context-independent approach, namely almost exclusive utilization of context-dependent operators, is also frequently employed in the design of artificial intelligence-machines. Such maximum a priori inquirers are represented by, for instance, Gelernter's[25] geometry proving program, Slagle's[26] symbolic integration program, and other special purpose performance machines.

In general, the design of such machines has been particularly successful in areas where the choice of representation seems more or less obvious to the human problem-solver as in the following examples:

a. The use of diagrams for geometry problems. Gelernter supplied a heuristic in the form of a coded model of a diagram to his geometry machine in order to provide a 'filter' for rejection of infeasible sentences generated as attempted sub-goals. "As an experiment, a number of attempts were made to prove extremely simple theorems with the latter heuristic 'disconnected' from the system (i.e., all non-circular subgoals generated were accepted)....We estimate conservatively that, on the average, a number of the order of 1000 sub-goals are generated per stage by the decoupled system. If one compares the latter figure with the average of 5 sub-goals per stage accepted when the diagram is consulted by the machine, it is easy to see that the use of a diagram is crucial for our system. (Note that the total number of sub-goals appearing on the problem-solving graph grows exponentially with the number accepted per stage)."[27] To be fair the result would, of course, be no less spectacular if the constraints were introduced into the problem by some other method. The diagram simply was one convenient way of doing so.

b. Family relationships by convention are represented in trees, so it is quite natural that Lindsay's SAD-SAM displays family relationships in the form of tree structures (a representation particularly suitable as the programming-language used was IPL-V).

c Fitting of lines to given sets of data is most easily visualized in a diagram. This method of representation has therefore been programmed into our sequence-extrapolators.

In terms of the representation of Figure 3.2, this approach pushes the line A as close to the solution as possible.

There is also a possibility of employing artificial intelligence-machines such as GPS[28] where a context-independent "permanent core" is embedded in a context-dependent "environment-machine," i.e., problem-solving is performed in a dialogue between two essentially separately functioning sub-machines The border between the domains of the sub-machines is in Figure 3.2 represented by the line A .

By choosing a narrow domain for a context-dependent inquirer, it may be possible to design quite powerful input-sectors, but the appeal of having a wide area of applicability is lost. If a wide domain is chosen the input-sector is likely to become poorly selective and, in short, mediocre in elegance and efficiency. The only way out of the dilemma seems to be the application of several powerful models which, when combined, cover a wide domain, but when correctly chosen permit strong selectivity. Such multiple maximum a priori Kantian inquiring-systems as exemplified by the complex sequence-extrapolator SEP, will be discussed in Section 5. The main components of such a system are indicated in Figure 3.3.

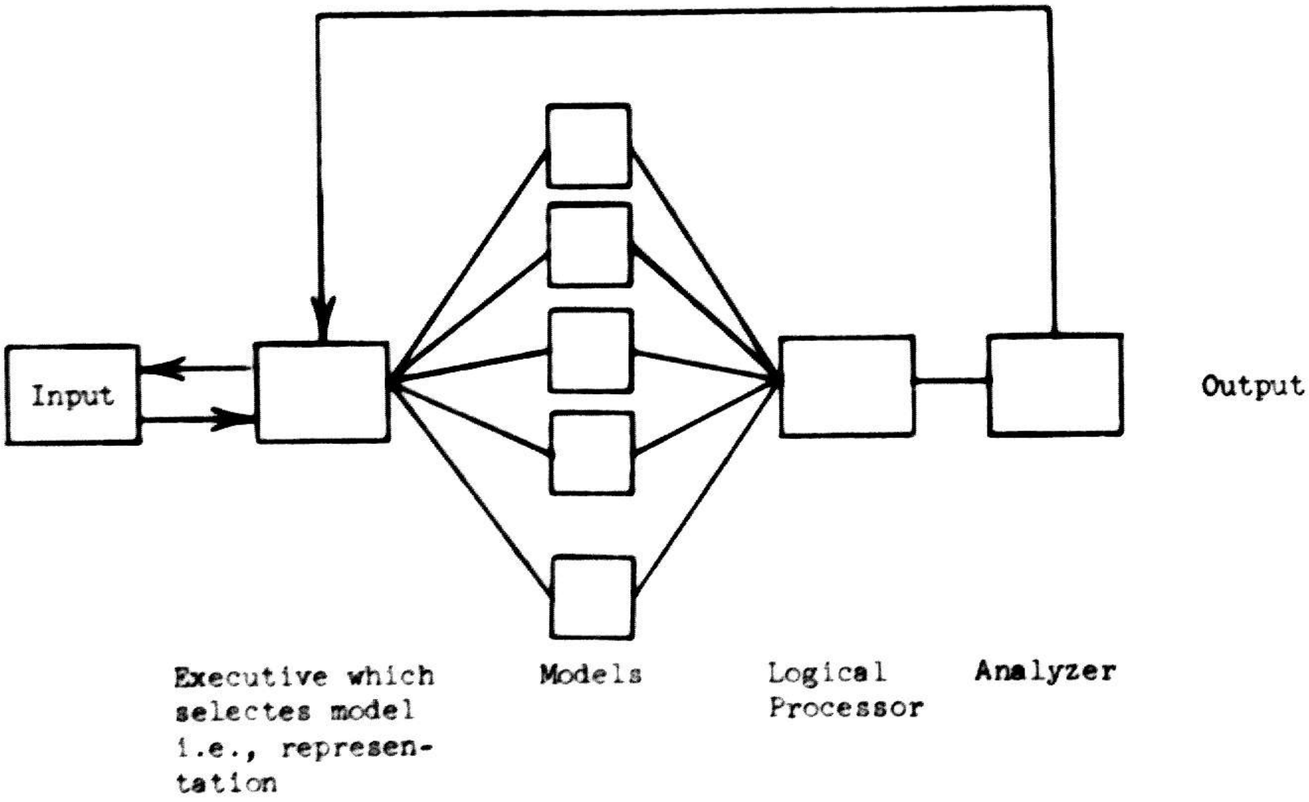


Figure 3.3

In Figure 3.3 there is a set of models, which define representations suitable for specialized domains of inquiry. When a problem arrives, an executive after studying the raw input-data decides which model to apply first. Input-data is represented according to the specifications of the chosen model, and the processing is performed. If the result is deemed successful, an output results. Otherwise the process is repeated, i.e., a new model is chosen, etc., The rules for choosing models, etc., are discussed in Section 5.

3.1.1 The Domino Problem

Newell[29] has presented the domino-problem as an example of the insufficiency of the stock of ideas concerning representation. McCarthy[30] has given a formal representation of the problem in the first order predicate calculus as "A Tough Nut for Proof Procedures." The problem is interesting because, if represented as a straightforward formalization, it is a very tough challenge for a logical processor. However, when represented in an efficient model, the solution of the problem is easily found.

The Problem: Is it possible to cover the mutilated checkerboard shown in Figure 3.4 by dominoes of size 1 × 2 squares?

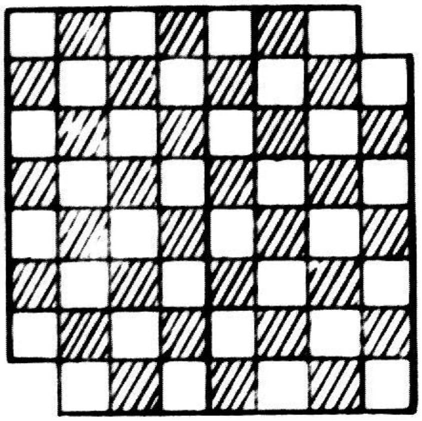


Figure 3.4

Newell,[31] in reviewing several different representations of the problem, states:

> Hence, the ultimate problem is not to discover the proof, but to build a machine that can discover the proof to the domino problem. It is a fair statement, I believe, that no one today knows how to build such a machine - or equivalently, how to construct such a computer program.

The statement does not fully represent Newell's and other writers' conception of the problem; the following quotation from a later section, however, provides a clarification:

> For, I believe. certainly, that given a modest amount of additional effort, a reasonable program can be constructed that finds the domino proof and does so fairly.

The last word should be emphasized! What a priori knowledge can be considered fair for an artificial intelligence-machine?

The question of "fairness" is recurrent in discussions of artificial intelligence. Mainly, this is because the total amount of contextual information available to a machine must be far smaller than that available for a human problem-solver, thus making addition of any information of pertinance to the context of a particular problem to stand out as "unfairly" given. Therefore, fairness seems to imply that no information present in the memory of a machine constitutes an anticipation of a particular problem or of a narrow class of problems. Thus, in the case of the domino-problem, such contextual information as the existence of black and white squares seems to be tabu (compare McCarthy's 'clean' representation, and Newell's careful avoidance of the subject) and we feel rightly so. But fairness must imply some contextual information, some methods which 'might' work, etc.

It should be observed that the question of fairness looses its significance if reference is made to careful definitions of the domains of inquirers, instead of imposing rules of fairness on the employed methods of solution.

The following proposal for a machine solution of the "Domino Problem" freely draws upon a priori knowledge, but still it should be considered fair in the light of our preceding discussion.

..

The 'solution will be given step by step. Pl is the initial problem, P2 and P3 are questions generated by the machine.

Pl. Is it possible to cover the mutilated checkerboard by dominoes?

P2. How can "to cover 'something' by dominoes" be represented?

P3. How can "to cover by a domino" be represented?

P3 represents the crucial step in our approach and criticism against it may certainly be justified because it seems to be so easily identified. It should, however, be noted that several different approaches are attempted in any 'real' situation, and that the 'one shot' success illustrated here may be the result of lengthy search.

Some a priori knowledge is required to represent "a cover;" as in the minimal a priori Kantian inquirer, a coordinate-system for space-relations is assumed to be available.

A representation of 'a cover of a square (I, J) is given in Figure 3.5.

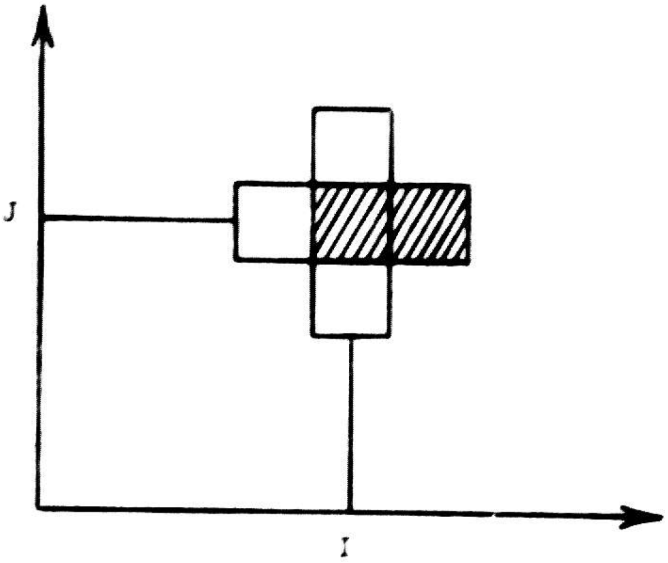


Figure 3 5

As seen, a 'cover' of the square (I, J) can be represented as any one of four pairs:

$$(I, J), (I + 1, J)$$

$$(I, J), (I - 1, J)$$

$$(I, J), (I, J + 1)$$

or $$(I, J), (I, J - 1).$$

The inquirer attempts to find simple representations of concepts; therefore, its "generalization sector" is employed to find a representation that subsumes the above four pairs. Looking for differences in the pairs, it is found that they only differ by +1 or -1, which suggests the picture of Figure 3.6

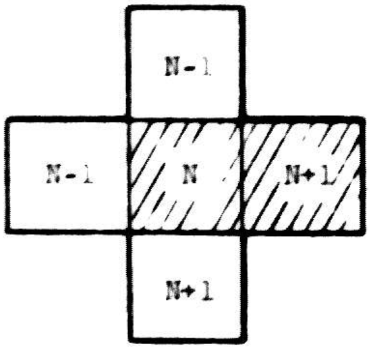


Figure 3.6

The representation of a cover is thus reduced to any one of two pairs:

$$(N, N + 1)$$

or $$(N, N - 1).$$

In the next step, the "generalization sector" attempts to combine these two pairs into a common representation. Such representation can be found as each pair consists of one odd and one even number.

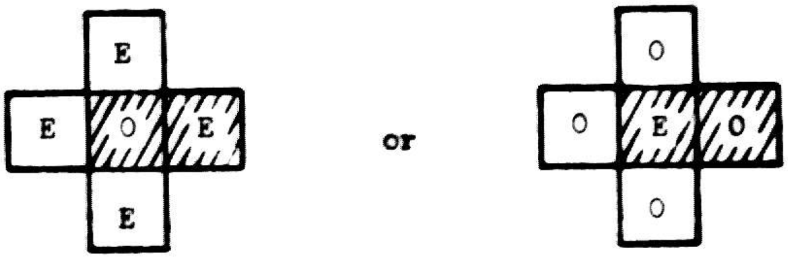


Figure 3.7

The representation of a cover then is (E, O) .

The translations between representations given in Figures 3.5, 3.6, and 3.7 are summarized in figure 3.8.

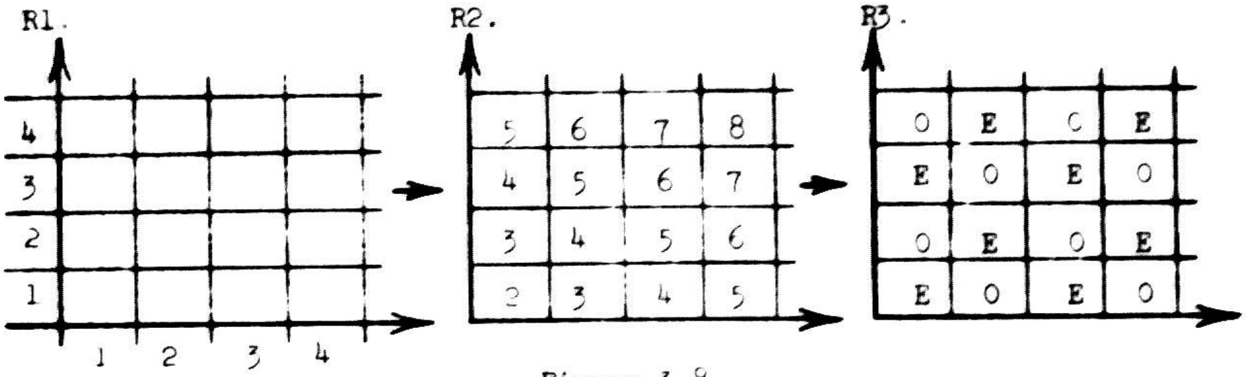


Figure 3.8

SOLUTION OF P3

Represent a 'cover' as (E, O) .

The next step is to find the solution of P2, i.e., to represent a board covered by dominoes.

This step requires a priori knowledge of the principle of mathematical induction, or, in Newell's words, "...If P(n) implies P(n + 1), and P(1) is true, then P(n) is true for all positive n . Now, there is only one such principle,.... Consequently, it is reasonable to assume that a problem-solving program would be given this principle." Purely formal operation of this principle gives the result that to cover something by n dominoes requires n pairs (E, O) .

SOLUTION OF P2

"To cover the mutilated checkerboard by dominoes" is represented as:

$$n \text{ pairs } (E, O) .$$

The next step is to find the solution of P1, i.e., to establish if the mutilated checkerboard can be covered by n pairs (E, O) .

1. The board has 62 squares, i.e., it can be divided into 31 pairs.

2. There are 30 E's and 32 O's, i.e., there cannot be formed 31 pairs of E's and O's .

SOLUTION OF P1

No, the mutilated checkerboard cannot be covered by n pairs (E, O) .

The described procedure explicitly searches for a simple representation of a cover. Even if the reader does not consider it to be a 'fair' solution, we hope that the simple lesson, "search for a simple representation," will justify the use of a well-known example on the importance of finding a representation. It should be observed that the simple induction-rules employed here also are employed in one of our sequence-extrapolation programs (see Section 4) and, therefore are not specific to the current example. This may increase the likelihood of a verdict of fairness for the presented solution.

3.2 Levels of Representation in Mechanized Models of Inquiry

The discussion of Section 3.1 indicates that some way of representing representations is needed in order to find a suitable representation of a problem. This result is not empty, but on the contrary, it clearly indicates the possibility of employing a hierarchy of representations, or languages, in the problem-solving process.

Such hierarchy extends all the way from representation of problems, via formulation of problem-solving programs down to the internal processing level of a digital computer. By visualizing problem-solving as a tree sorting procedure we (because of the recursive definition of a tree)[32] implicitly suggest that search for representations be performed at any level of the process.

Therefore, an initially considered problem-formulation may, during the process of problem-solving, be translated into other representations in order to permit generalization and/or application of particular techniques. The former case will now be briefly discussed.

"The ability to discern similarities beneath divergence is the ability to generalize," according to the psychologist Humphrey.[33] Our task is to find representations that permit such generalizations to be made. There is an abundant literature on inductive logic, but still it is very difficult to find some description of a language suitable for representing generalization. A quotation from E. Cassirer[34] characterizes the situation,

> Thus, abstraction is very easy for the "philosopher" but, on the other hand, the determination of the particular from the universal, so much more difficult.

There are a few examples of application of inductive methods for particular situations represented in the literature on artificial intelligence. London[35] used induction on length of strings as a practical tool. But his method is based on a particular theorem-proving situation and he consequently states,

> The mechanization of these techniques [induction and case analysis] is tied intimately to this task and their separation from the task along the lines of GPS is desirable if the program is to become a more general aid and theorem-prover.

Solomonoff[36] describes a grammatical induction scheme. A very simple but general induction technique is employed in the exemplification of the domino problem (see Figure 3.8) for representing a 'cover,' where a specific representation (R1) is first slightly generalized (R2) and eventually a still more general representation (R3) is used for the remainder of the process. Another example is provided by our sequence-extrapolating programs, where the input is represented as a sequence of numbers which is to be modeled. In cases where no a priori models are available, the task is fairly straight-forward, but, when the program itself builds models, then the representation must be such that it permits generalization of experienced results. This requirement will be discussed in Section 4.

An instructive example of application of multi-level languages is given by T. Evans[37] in his discussion of a program for analysis of pictorial analogies.

By utilizing multiple levels of language, situations (which at one level of abstraction seem to be unrelated) may, at a higher level, be naturally connected. Occurrences which, in one representation, seem random are clearly causal at a different degree of abstraction. Multiple languages permit abstraction, specialization, utilization of different relational models on a constant data-base and, in short, gives the "linguistic flexibility" necessary for sophisticated problem-solving.

3.3 Pragmatics of Representation

The importance of suitable labelling, as discussed in the context of Lockean inquiring-systems, applies equally well to mechanical inquirers. The following pair of quotations illustrates the situation. The psychologist K. J. W. Craik[38] writes:

> The effects of language in perception appear to be to make those features of the objective world that are represented by linguistic forms stand out in greater articulation.

This quotation is strongly related to the computer scientist, Minsky's,[39] statement:

> It is usually necessary to have ways of assigning names (symbolic expressions) to the defined classes. The structure of the names will have a crucial influence on the mental world of the machine because it determines what things can be conveniently thought about.

We have shown how EPAM from given representations of complex entities abstracts features, which suffice for an efficient parsimonious labelling and, also, how the noticing-order employed by EPAM will affect its categorizations. In general, computer-generated labels can be made more suited for logical processing and classification than the conventional names of natural language are. In particular, 'content based' addressing, as in EPAM, can be efficiently implemented.

In the pioneering works of Newell, Shaw, and Simon,[40] there is a recurrent realization that sophisticated problem-solving requires powerful means of representing models. They write:

> There is a close and reciprocal relation between complexity and communication. On the one hand the complexity of the systems we can specify depends upon the language in which we must specify them....

and

> a more powerful language can specify greater complexity with limited processing powers.

There is an obvious trend toward development of increasingly sophisticated programming languages. We can trace how the basic signal language of bits and gates of a computer is activated from successively more complex languages such as machine-language, assembly-language, problem-oriented languages,[41] and all the way up to sub-routines and executive-routines of problem-solving programs. Our main concern, however, has been to study the languages from the other end of the programming process, i.e., we have discussed the languages relevant to problem-solving. As there is, however, a very definite interaction between computer-languages and formulations of problem-solving machines, not only in components but also in their very structure, a few comments on the situation will be made. The users of the programming language, LISP, for instance, sometimes "get carried away" by its convenient recursion facilities[42] or choose problem-environments particularly suited to its mathematical structure. Equally obvious is the tendency to structure problems according to IPL V's moı equential structure. Raphael[43] has, in somewhat confused terms, tried to classify computer-languages as declarative or imperative where the former type (if implemented) would be more suitable for formulation of programs of artificial intelligence. It should be observed here that "simulated" declarative languages can be devised by proper structuring of sub-routines. Although very interesting, the subject of pragmatics of programming-languages will not be discussed further in this context.[44]

FOOTNOTES AND REFERENCES FOR SECTION 3

1. It is assumed that we know that there is a problem.

2. Henri Bergson, "Introduction to Metaphysics," 1912.

3. Susan Langer, "Philosophy in a New Key," Harvard University Press, 1942.

4. Bruner, Goodnow, and Austin, "A Study of Thinking," John Wiley and Sons, Inc., New York, 1956.

5. G. Humphrey, "Thinking," London Methuen, 1951.

6. A. D. Ritchie, "The Natural History of Mind," Langmans Green and Co., 1936.

7. Dean E. Wooldridge, "The Machinery of the Brain," New York, McGraw-Hill, 1963.

8. The digital computer is a cause-effect system, however, the availability of delay-elements permits design of sophisticated models.

9. This essentially requires almost isomorphic representations.

10. B. L. Whorf, "Language, Thought, and Reality," New York, John Wiley and Sons, Inc., 1956.

11. For a similar view, see Vernon, M. D., "A Further Study of Perception," Cambridge, Cambridge University Press.

12. L. von Bertalanffy, "An Outline of General Systems Theory," British Journal of Philosophical Science, Aug., 1956.

13. D. M Mackay, "Quantal Aspects of Scientific Information," Phila. Mag. 41, 1950.

14. J. E. Mulligan, "Basic Optimization Techniques - A Brief Survey," Journal of Industrial Engineering, May-June, 1965.

15. C. West Churchman, "Kantian Inquiring Systems," Internal Working Paper No. 46, Space Sciences Laboratory, Social Sciences Project, University of California, Berkeley, May, 1966.

16. G. Polya, "Mathematical Discovery," New York, John Wiley and Sons, Inc., 1962

17. See almost any paper reprinted in E.A. Feigenbaum and J. Feldman, editors, "Computers and Thought," New York, McGraw-Hill, 1963.

18. R. K. Lindsay, "Inferential Memory as the Basis of Machines Which Understand Natural Language," reprinted in ref. [17].

19. J. R. Slagle, "Experiments with a Deductive Question-Answering Program," Communications of the ACM, Vol. 8, No. 12, Dec., 1965.

20. A. Newell, J. C. Shaw, and H. A. Simon, "Empirical Explorations With the Logic Theory Machine - A Case Study in Heuristics," reprinted in ref. [17].

21. J. McCarthy, "Programs With Common Sense," in "Mechanization of Thought Processes," Vol. I, Proc. Symp. Natl. Physical Lab., 24, 25, 26, and 27, November 1958.

22. B. Raphael, "A Computer Program Which Understands," in Proc. of the Fall Joint Computer Conf., Spartan Books, 1964.

23. D. G. Bobrow, "A Question-Answering System for High School Algebra Word Problems," in Proc. Fall Joint Computer Conf., Spartan Books, 1964.

24. A rather uncritical survey of fifteen experimental English language question-answering systems is given in R. F. Simmons, "Answering English Questions by Computer: A Survey," in Communications of the ACM, Vol. 8, No. 1, Jan., 1965.

25. H. Gelernter, "Realization of a Geometry-Theorem Proving Machine," reprinted in ref. [17].

26. J. R. Slagle, "A Heuristic Program That Solves Symbolic Integration Problems in Freshman Calculus," reprinted in ref. [17].

27. H. Gelernter, J. R. Hansen, and D. W. Loveland, "Empirical Explorations of the Geometry-Theorem Proving Machine," reprinted in ref. [17].

28. A. Newell and H. A. Simon, "GPS, A Program That Simulates Human Thought," reprinted in ref. [17].

29. A. Newell, "Limitations of the Current Stock of Ideas About Problem-Solving," in "Electronic Information Handling," edited by A. Kent and O. E. Taulbee, Wash., Spartan Books, 1965.

30. J. McCarthy, "A Tough Nut for Proof Procedures," Stanford Artificial Intelligence Project, Memo No. 16, 1964.

31. See ref. [29].

32. A tree is an ordered n-tuple, the elements of which themselves may be ordered n-tuples (with, in general, different values of n) and so on, to any depth.

33. See ref. [5].

34. E. Cassirer, "Substance and Function," New York, Dover Pub., 1953.

35. R. L. London, "A Computer Program for Discovering and Proving Sequential Recognition Rules for Well-Formed-Formulas Defined by a Bachus Normal Form Grammar," thesis, Pittsburgh, Carnegie Institute of Technology, 1964.

36. R. J. Solomonoff, "An Inductive Inference Machine," in IRE Natl. Conv. Record, Part 2, 1957.

37. T. Evans, "A Heuristic Program to Solve Geometric-Analogy Problems," in Proc. AFIPS, 1964, Spring Joint Computer Conf., Spartan Books.

38. K. J. W. Craik, "The Nature of Explanation," Cambridge, Cambridge University Press, 1943.

39. M. Minsky, "Steps Toward Artificial Intelligence," reprinted in ref. [17].

40. See ref. [17].

41. Such as compilers and interpreters.

42. A typical example is provided by M. Pivar and M. Finkelstein, "Automation Using LISP, of Inductive Inference on Sequences," in "The Programming Language LISP," edited by Berkeley and Bobrow, Cambridge, Mass., 1964.

43. B. Raphael, "The Structure of Programming Languages," discussed in Communications of the ACM, Vol. 9, No. 3, March, 1966, and No. 4, April, 1966.

44. An overview of the field is given in "Proceedings of the ACM Programming Languages and Pragmatics Conference, San Dimas, California, Aug. 8-12, 1965," in Communications of the ACM, Vol. 9, No. 3, March, 1966.

BLANK PAGE

4. Sequence-Extrapolation

Sequence-extrapolation has been chosen as the domain of a computer-model of a representational inquiring-system which will be presented in the next few sections. As representational functions of inquiring-systems are strongly context-oriented (Section 3.1.1), a brief introduction to the field of sequence-extrapolation will be given in order to provide background for later discussion.

Sequence-extrapolation, or inquiry into the structure of sequential patterns, is a process of establishing relationships, rules of progression, between members of a series of symbols. Such inquiry is deemed successful, i.e., knowledge has occurred when sets of identified rules, i.e., models, can be applied to generate arbitrary members of corresponding sequences.

At first glance, sequence-extrapolation will seem to require application of genuine induction, i.e., to start out from a pattern, represented by an input-sequence, and eventually arrive at a more general representation from which the input-sequence may be deduced. However, true indùctive reasoning is not necessarily required. In many cases, apparent inductive behavior should rather be described as "deduction disguised as induction".[1] Some such cases will be indicated during the following discussion.

The process of establishing relationships between members of a sequence is analogous to finding grammars for sets of sentences. Such grammars can be trivially designed by enumeration or by employing rules such as: "sentence → sentence + any word". Any apparent generality of such grammar is necessarily not sufficient reason to accept it as a desired type of description. The easily realized drawbacks of these "general grammars" are not as easily discovered when translated to the analogous case of sequence extrapolation. In particular, some confusion about this situation has been noticeable in published work on sequence-extrapolation. Specifically, the question of power, or inductive power, of sequence-extrapolators seems to have been misunderstood. The next section will therefore be devoted to this subject.

4.1 "Generality" in Sequence-Extrapolation

In general, there exists no unique continuation to any sequence of symbols. Actually, no continuation can be ruled out as infeasible.

EXAMPLE 1. Which are the next few entries of the sequence:

1, 2, 3, 4, 5, ... ?

As for any sequence, there are infinitely many possible continuations, some of which are:

6, 7, 8, 9, 10, ... (a)
1, 2, 3, 4, 5, ... (b)
8, 7, 16, 9, 32, ... (c)
7, 9, 11, 13, 16, ... (d)

A few of the patterns may seem rather unlikely, but they can all be justified if seen in their proper context:

a) This is the continuation which would be preferred by most knowledgeable people if no particular context is defined, i.e., in the typical case where the human problem-solver knows the order of the numbers and sees no reason that the progression will not continue as started.

b) This pattern gives a cyclical repetition of the subsequence 1, 2, 3, 4, 5 . It may be chosen under the influence of previous exposure to cyclical patterns, e.g., calendar data.

c) This sequence has actually been given by a sequence-extrapolation program. The situation was such that the program had encountered several patterns consisting of intertwined subsequences, i.e., the pattern was assumed to be:

$$1,\ \underline{2},\ 3,\ \underline{4},\ 5,\ \underline{8},\ 7,\ \underline{16},\ 9,\ \underline{32},\ \text{etc.}$$

d) This series is somewhat tricky. It consists of the primes and their powers arranged in order of magnitude.

The example is homely, but hopefully it helps to illustrate the point that the context within which a pattern is presented should have a major influence on any prediction of its continuation.

The non-uniqueness of continuations of sequences thus forces us to consider discovery as well as evaluation-procedures for representations of sequences.

Discussion of evaluation will be postponed. For the moment, it suffices to note that such evaluation is the task of the executive of Leibnitzian inquirers. Representation of sequential patterns, on the other hand, will be illustrated in the context of a few simple examples.

Given a sequence of n numbers, we can always find a polynomial $C_{n-1}X^{n-1} + C_{n-2}X^{n-2} + \dots + C_1X + C_o$ which for consecutive values of X, $X = 1,2,\dots, n$; will represent the initial sequence.[2] Thus, any sequence of n numbers can always be described by an expression containing n independent parameters which cannot only recreate the sequence but also provide one justifiable continuation. Such extrapolator is described in Section 6.1. This extrapolator will hereafter be denoted as E1 E1 is completely general, but will not always provide representations which seem reasonable.

Certain shortcomings of a class of "general" sequence-extrapolators[3] will be demonstrated by exemplification

<u>EXAMPLE 2</u>[4] 1, 10, 100, 1000, 10000, <u>40951</u>

The extrapolation 40951 does not seem reasonable. By representing the input sequence differently, however, E1, as shown in Example 3, is applicable

<u>EXAMPLE 3</u> 10^0, 10^1, 10^2, 10^3, 10^4, $\underline{10^5}$

The extrapolation is here performed on the sequence of exponents.

A generalized version of E1, described in Section 6.3, here denoted E2, has a somewhat richer domain of representations.

<u>EXAMPLE 4</u> 1, 10, 100, 1000, 10000, <u>100000</u>

The result is derived via an implicit translation of the input sequence to n_i n_{i-1} 10, $n_1 = 1$, where n_i is the nth element of the sequence[5]

<u>EXAMPLE 5</u> 1, 11, 111, 1111, 11111, <u>111111</u>

Here E2 finds $n_i = (n_{i-1}\ 10) + 1$, $n_1 = 1$.

<u>EXAMPLE 6</u> 1, 12, 123, 1234, 12345, <u>123456</u>

F2 translates to $n_i = (n_{i-1}\ 10) + i$, $n_1 = 1$.

In general, E2 applies over a domain described by $n_i = P^1(n_{i-1}, i)$, where P^1 denotes a linear combination of its arguments.

<u>EXAMPLE 7</u>. 1, 22, 333, 4444, <u>15865</u>

Here E2 obviously fails while a simple representation which could easily be made internal to E2 succeeds. Compare Example 8.

<u>EXAMPLE 8</u>. 1, 22, 333, 4444, <u>55555</u>

where $n_i = i \cdot (\frac{n_{i-1}}{i-1} \cdot 10 + 1)$, $n_1 = 1$ or the sequence

$1 \cdot 1, 2 \cdot 11, 3 \cdot 111, 4 \cdot 111, \ldots, 10 \cdot 1111111111$

or where $n_i = i:i$, where : stands for number of occurrences giving the sequence 1:1, 2:2, 3:3, 4:4, 5:5, ..., 10:10 .

(Note the dissimilar results for $i = 10$, i.e., 11111111110 and 10101010101010101010, respectively.)

The examples presented are very simple but, hopefully, they have conveyed the idea that even if we add features which take some of the shortcomings out of a "general" sequence-extrapolator, there will still be unlimited numbers of sequences which will be extrapolated in a very awkward manner. Obviously, context must be considered and, within any context, simplicity and elegance should also be taken into account.

Thus, generality is not necessarily a virtue of sequence-extrapolators but may actually be a hindrance for reasonable performance. Therefore, the subject of context in sequence-extrapolation will be discussed in the next section.

4.2 <u>Context in Sequence-extrapolation</u>

"How can you do 'new math' problems with an 'old math' mind?"

(Linus in Peanuts by Charles Schultz)

Why do certain extrapolations seem more "right" than others?

How should an evaluation-procedure for sequence-extrapolation be designed?

These questions are very difficult to answer with authority as correctness in sequence-extrapolation is clearly a function of how well the results fit into the particular context within which the problem is formulated. Even if the context is well-defined, there still remains to evaluate alternatives which clearly cannot be judged on the criterion of relevancy for context only. Thus, the well known but rarely objectively defined notions of simplicity, elegance, parsimony, etc., will have to be employed. The only guide that can be given seems to be that the most appealing of those alternatives which fit into the context should be chosen. This guide is admittedly weak but still it is preferable to an uncritical acceptance of the first alternative that comes along.

The importance of context has been expressed repeatedly but how is it introduced into mechanical sequence extrapolators?

There are several methods available and we will now proceed to exemplify those which are most frequently employed.

1. Design the extrapolator to apply a few specialized methods (see 3 below) but, if these fail, to resort to some general "clean up" method. This approach is used in an extrapolator by M. Pivar and M. Finkelstein,[6] which conceptually is very close to the previously described E2. They claim:

> This report deals with the problem of programming a computer to perform inductions on certain general kinds of data in a manner superior to the majority of human beings.

In view of the poor selectivity of the program, the above passage clearly confuses uncritical generality, as demonstrated in Section 4.1, with inductive power. However, the risk of confusing "inductive power" with efficient algorithms for exploring very narrow domains must also be realized.

Simon and Kotovsky[7] have published an account of a specialized sequence-extrapolator for the Thurstone letter-completion-tests. Although parsimonious in respect to immediate memory requirement and variety of operators, the extrapolator can satisfactorily account for most of the test-sequences presented in their paper. However, the authors' claim that "several versions of the program show varying levels of inductive power" seems to miss the point that the purely deductive Leibnitzian inquirer they present only needs a very limited amount of trial and error or exhaustive search to compile a generating model for any valid sequence.

A more straight-forward procedure, covering the same domain, is presented in Section 4.2.2, where an extrapolator is described which can efficiently extrapolate sequences from a very well-defined domain.

4.2.1 A Sequence-extrapolator Which Builds a Model of Its Domain

An extrapolator which, using a simple meta-language, can perform certain elementary generalizations is developed. The particular model is rather restricted in scope but the methods employed are of wider applicability. The inquirer is divided into two parts:

A. a model-building and evaluating executive, and

B. a set of tools for construction of models.

The executive is requested to build a model of sequences which, during a learning period, are presented for analysis and extrapolation. The executive is general, in the sense that its methods are not context-dependent.

The executive operates on r-tuples which are manipulated by simple rules of induction. These rules are:

1. A set of n identical symbols S may be represented as a pair (n, S).

2. A r-tuple of symbols which belong to the same category C may be abstracted to a pair (r, C).

3. Initial states of sequences may be separated from the relational structure.

Part B of the program is context-dependent. The present discussion will concern itself with a domain defined by a particular type of letter-sequences which are used in Thurstone Letter Completion Tests[8]; however, several other domains can easily be implemented.

The following a priori facilities are available:

a. an alphabet, (usually the English);

b. facilities to establish if any one of a set R of relations holds

$$R = \{ =, +, - \}$$

$(=)$ = equality

$(+)$ = successor

$(-)$ = predecessor ;

c. facilities to establish cyclicity.

Cyclicity is established by the shifted difference approach, i.e., a sequence $Y = y_1, y_2, \ldots, y_i, \ldots, y_n$ is cyclical with the period k, if for all i, $y_i = y_{i-k}$ Cyclicity can be represented as

$$\begin{array}{l} y_1, \ldots, y_{1+k}, \ldots, y_{i+k}, y_{i+k+1}, y_{i+k+2}, \ldots, y_n \\ \underline{\qquad\qquad y_1, \ldots, y_i, y_{i+1}, y_{i+2}, \ldots, y_{n-k}, \ldots, y_n} \\ \qquad\qquad =, \ldots, =, =, =, \ldots, =, \end{array}$$

or $(=, =, \ldots, =)$

A description of how a particular learning-period will influence the model-building is given. The examples are taken from Simon and Kotovsky.[8]

A)

$$\begin{array}{lcccccccc}
 & a & b & a & b & a & b & & \\
k = 2 \rightarrow & & & a & b & a & b & a & b \\
\hline
 & & & = & = & = & = & &
\end{array}$$

It should be noted that a model is designed to be a symbolical representation of a relational structure; thus, where possible, details of components are excluded. Hence, by induction rule 3, i.e., in analogy with mechanical models, the initial state is separated from the relational model. The sequence is conveniently described as:

$$\underbrace{(a,\ b)}_{\text{Initial State}} \qquad \underbrace{(=,\ =)}_{\text{Model } M_A}$$

By rule 1 the model may be written $(2, =)$.

B)

$$\overset{\neq}{\overleftrightarrow{c \quad a \quad d}} \quad a \quad e \quad a \quad f \quad a$$

As indicated by the arrow, model M_A doe not apply. A more detailed study is therefore required. There are several different approaches available, two of which will be shown.

$$\begin{array}{lcccccccccc} 1. & c & a & d & a & e & a & f & a & & \\ 2. & & & c & a & d & a & e & a & f & a \\ \cline{2-11} 3. & & & \neq & = & \neq & = & \neq & = & & \end{array}$$

The sequence of line 3 does not indicate cyclicity, unless a generalization of the cyclicity-operator can be found. A potential generalization is:

$$(c, a)(R, =) .$$

The nature of R will have to be studied. It turns out that R corresponds to (+), which suggests the model M_B .

$$M_B : (c, a)(+, =) .$$

Here a generalization has been made; instead of the parameter (=) of the cycle-operator, the operator (+) is employed. A *similar* parameter has been found. Similarity is, in the present model, defined as shared class-belonging, however, (=) and (+) both belong to the class of relations $R = \{ =, +, - \}$ and, therefore, (+) is accepted in the present situation.

Another approach is to apply the cyclicity-operator on a model of the relational structure of the input-sequence.

$$\begin{array}{lcccccccccc}
\text{Input} \rightarrow & c & a & d & a & e & a & f & a & & \\
 & & & c & a & d & a & e & a & f & a \\
\cline{2-11}
\text{Model} \rightarrow & & & \neq & = & \neq & = & \neq & = & & \\
 & & & & & \neq & = & \neq & = & \neq & = \\
\cline{2-11}
 & & & & & = & = & = & = & &
\end{array}$$

The model sequence can be described as:

$$\underbrace{(\neq,\ =)}_{\text{Initial State}} \qquad \underbrace{(=,\ =)}_{\text{Relations}}$$

Use the model to "mask" out the "questionable" sequence.

$$\begin{array}{rcccccccc}
\text{Input} & c & a & d & a & e & a & f & a \\
\text{Model} & \neq & = & \neq & = & \neq & = & \neq & = \\
\cline{2-9}
\text{Sequence} & c & & d & & e & & f &
\end{array}$$

This is a simple successor-sequence which, by the _a priori_ facilities, can be recognized as:

$$(c)(+) \ .$$

Substituting $(c)(+)$ for $(\neq)(=)$ gives $(c,\ a)(+,\ =)$ i.e., Model M_B .

C) $\quad a \quad a \quad b \quad b \quad c \quad c \quad d$

No cyclicity can be directly applied, but reverting to a model of the sequence permits analysis.

	a	a	b	b	c	c	d		
			a	a	b	b	c	c	d
Relational model			+	+	+	+	+		

A generalization, as in Example B, provides a model;

$$M_C : (a,\ a)(+,\ +)\ .$$

The value of $k = 2$ is the result of previous experience and, furthermore, other periods do not provide models compatible with the periodicity of the input-sequence.

The presently employed models are:

$$M_A : (=,\ =)$$

$$M_B : (+,\ =)$$

$$M_C : (+,\ +)$$

Completeness suggests:

$$M_X : (=,\ +)\ .$$

The relation $(-)$ is a potential candidate for entering the models. By rule 2 the models can be generalized to:

$$M : (u, v) \qquad u, v \in R$$

D) $\quad a \quad b \quad x \quad c \quad d \quad x \quad e \quad f \quad x \quad g \quad h \quad x$

The present model cannot describe this sequence; it will, therefore, have to be filed away for future analysis.[11]

E) $\quad \overrightarrow{a \quad x \quad b \quad y} \quad a \quad x \quad b \quad y \quad a \quad x \quad b$

At first glance, the model $(a, x)(+, +)$ seems applicable, however, further testing establishes its falsity. The general model M does not apply and a complete investigation will have to be made.

$$\begin{array}{cccccccccccccccc}
 & a & x & b & y & a & x & b & y & a & x & b & & & & \\
k = 4 & \rightarrow & & & & a & x & b & y & a & x & b & y & a & x & b \\
\hline
 & & & & & = & = & = & = & = & = & = & & & &
\end{array}$$

$$M_E : (a, x, b, y)(=, =, =, =) .$$

The model resembles the previously developed ones and a consolidation of models is conceivable.

$$M : (u, v) \qquad u, v \in R$$

$$M : (s, t, u, v) \qquad s, t, u, v \in R$$

The second model is derived from M_E as a generalization by rule 2 .

R has been substituted for $(=)$. The models may be represented as pairs:

$$M : (2, r) \qquad r \in R$$
$$M : (4, r) \qquad r \in R .$$

A simple unified representation would be:

$$M : (m, r) \qquad m \in \{2, 4\}, \; r \in R .$$

The range of m can be tested by random generation of sequences which are presented to a guarantor for approval or disapproval. A further generalization, which is quite feasible and actually can be easily derived from a suitable example, is to expand the set of relations to $\{=, +, -, M\}$, i.e., by adding the model itself to the set of relations.[12]

Supposing that a period of 2 has been very frequently encountered, i.e. a large fact-net (see Section 2.1) reinforces the period 2, then an attempt to employ the following representation is made:

$$M^* : (a, x)(R1, R2) .$$

The sequences corresponding to R1 and R2 are:

$$\begin{aligned} & R1 : a, b, a, b, a, b \\ & R2 : x, y, x, y, x, \_ \\ \text{i.e.,} \quad & R1 : (a, b)(=, =) \\ & R2 : (x, y)(=, =) . \end{aligned}$$

This suggests

$$M^* : ((a, b), (x, y)) ((=, =), (=, =)) .$$

Thus M is, as suggested above, included in the set of relations.

F) r s r t r u r v r

The model M_F is easily found:

$$M_F : (r, s)(=, +)$$
$$M_F \in M .$$

G) a b c d a b c e a b c f a b c

$$M_G : (a, b, c, d)(=, =, =, +) .$$

This reinforces M as previously only (=, =, =, =) of length 4 has occurred.

H) m n l n k n j n

$$M_4 : (m, n)(-, =) .$$
$$M_4 \in M .$$

This is the first occurrence of (-) .

I) m n o m o o m p o m

$$(m, n, o)(=, +, =) .$$

This suggests that the domain of m should be investigated. At present,

$$m \in (2, 3, 4) \ .$$

J) c e g e d e h e e e i p f e

$$M_J : (c, e, g, e)\ (+, =, +, =)$$

Or, in a different mode of expression, i.e., by considering M as a relation,

$$M_J^* : ((c, g), e)((+, +), =) \ .$$

The present model M describes a major part of the domain of letter completion tests. Although the model is developed by rather bold steps of generalization, the methodology is still justified because the sequence-extrapolator is conceived to be interrogated in time-sharing, where the executive can test its hypothesis by asking the interrogator for validation of sequences generated by proposed models.

4.2.2. A Specialized Model

When knowledge about the structure of a domain is achieved, e.g., when the model M of section 4.2.1 is developed, efficient problem-solving may be performed. In the present section, the domain of letter-completion-tests, as described by the model M of the preceding section, will be utilized for efficient sequence-extrapolation.

The model is:

$$(m, r) \qquad m \in \{2, 3, 4\}; \ r \in \{=, +, -, M\} .$$

By temporarily disregarding the relation M, a simple strategy for extrapolation can be designed.

Given an input-sequence $Y = y_1, y_2, \ldots, y_n$, extrapolation is performed in three steps.

1. Find the value of m .
2. Find the relation r between y_{n+1-2m} and y_{n+1-m} .
3. $\underline{y_{n+1}} \leftarrow r\ (y_{n+1-m})$.

The steps can most conveniently be represented graphically, as in Figure 4.1. Let the letters of the alphabet be represented on the ordinate-axis and the order of elements of a sequence along the abscissa of a 2-dimensional coordinate system.

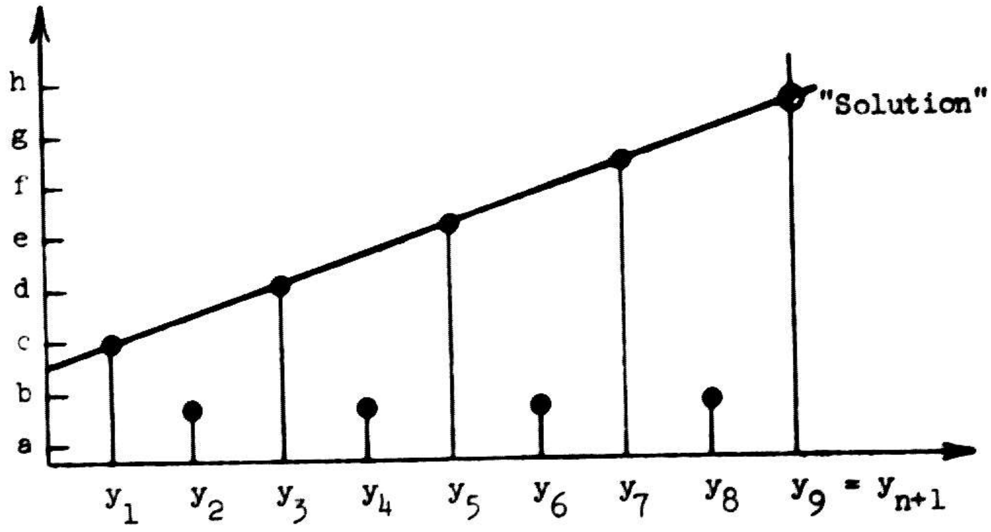


Figure 4.1

The procedure is as follows:

1. Starting at the point defined by entry y_{n+1-m}, where $m = 2$, i.e., in the example at y_7, test if a line of slope ± 0, $+1$, or -1 touches entry y_{n+1-2m}, i.e., in the example y_5. If this is the case, also check points y_{n+1-3m}, y_{n+1-4m}, etc., until the sequence is exhausted. In Figure 4.1, the negatively sloping line touches points y_7, y_5, y_3, and y_1. If such line does not exist, increment m by 1 and repeat the above steps. After, at most,[13] $(m - 1) \cdot 3 + \frac{n}{m} - 1$ elementary tests, where $2 \leq m \leq 4$, the periodicity of any valid sequence is established.

2. The relation r between y_{n+1-2m} and y_{n+1-m} is defined by the slope of the line establishing periodicity, i.e., no new test will have to be performed.

3. y_{n+1} can be computed from available information, i.e., by extending the line until the intersection with y_{n+1} is found.

As seen, an <u>exhaustive</u> <u>search</u> only requires $8 + \frac{n}{m}$ or, in general, approximately 12 elementary tests in the <u>worst</u> possible situation.

By optimizing the search-strategy of step 1, an average of $3 + \frac{n}{m}$ elementary tests can be expected to suffice for extrapolation. Thus, the required level of "inductive power" of sequence-extrapolation over this domain is very low.

4.3 <u>Error Correction in Sequence-extrapolation</u>

In a constrained environment, redundant information may be utilized for error-detection and/or error-correction in "nearly" valid input sequences. Although it is true that it is impossible to judge any sequence as invalid, in the Leibnitzian tradition the largest fact-net is also presumed to be the most reliable, and therefore, minor changes which permit a sequence to be connected to such a net are assumed to be permissible, in particular, if the input-mechanism is known to be unreliable.

Error-correction cannot be performed by "general sequence-extrapolators," thus El would unhesitatingly produce:

$$1, 2, 3, 4, 4, 6, 7, 8, 9, \underline{-116}$$

instead of reporting, "I believe the sequence to be"

$$1, 2, 3, 4, \underline{5}, 6, 7, 8, 9, \underline{10}\ .$$

Extrapolators which establish their domain by experience can, in many cases, reinforce models which have proved successful in a long run of problems such that "erroneous" sequences are not permitted to influence the design of the model. Furthermore, certain irregularities may be detected by very simple tests. The greatest potential for error-correction is, of course, to be found in strongly constrained sequence-extrapolators. A particularly powerful error-correction facility has been implemented in our sequence-extrapolator, SEP, where several types of errors can be dected and corrected. A brief description of these facilities will be given in order to illustrate the power of such design.

Section 6.1.1 describes a set of error-correction procedures derived for polynomial-extrapolation. By permitting certain transformations, which preserve the properties upon which the procedures are based, we can, however, stretch the validity to cover more general situations. In particular, monotonous transformations of polynomial sequences are still valid objects for error-correction. The following cases are satisfactorily handled.

Case 1. At most, E entries of the input-sequence Y are incorrect.

An estimate Y_s of Y is produced by applying the extrapolator to $m + 1$ consecutive values $(y_{0+j}, y_{1+j}, \ldots, y_{m+j})$ $j = 0, 1, \ldots, n-m$. If Y_s differs from Y in, at most, E places it is accepted, otherwise the procedure is repeated for new values of j until an acceptable estimate of Y is found.

Case 2. At most, E pairs of entries are interchanged. Apply the procedure of case 1 but permit differences between Y_s and Y in, at most, E pairs of positions such that by interchanging the members of such a pair corresponding differences are eliminated.

e.g., Y : 1, 2, 6, 4, 5, 3, 7, 8

Y_s: 1, 2, $\underline{3}$, 4, 5, $\underline{6}$, 7, 8

Combinations of case 1 and case 2 may, of course, be handled by the same procedures.

Case 3. At most, E entries of Y are missing.

The present case is "diagnosed" when Y_s and Y differ in leading or trailing strings of consecutive entries. By adding "dummy" members to the input-sequence, such that the number of discrepancies is minimized, identification of missing entries can be performed, e.g. Y : 1 2 3 5 6 7 8 10 11

Y_s : $\underline{2}$ $\underline{3}$ $\underline{4}$ 5 6 7 8 $\underline{9}$ $\underline{10}$

Y' : 1 2 3 D 5 6 7 8 D 10 11

Y'_s : 2 3 $\underline{4}$ 5 6 7 8 $\underline{9}$ 10

Dummies are placed where the runs of discrepancies start.

Case 4. The input-sequence Y is scrambled.

Section 6.1.1 gives a detailed account of how this case can be handled satisfactorily.

In general error-correction-procedures are based upon context-dependency and simplicity. The role of simplicity and efficiency in error-correction is rather interesting and will be briefly discussed.

The authors of "Automation, Using LISP, of Inductive Inference on Sequences,"[14] have proposed the following method for error-correction (detection of irregularities).

If, in a difference table[15] at some level k, the majority of the $\Delta^k_{y_i}$ are equal, it is assumed that all entries at this level should be equal. By working backwards, a modified difference-table defining an "ideal sequence," is constructed, e.g.,

Y 1 2 3 4 5 5 7 8 1 2 3 4 5 6 7 8

$\Delta^k Y$ 1 1 1 1 0 2 1 → 1 1 1 1 1 1 1

The "ideal sequence" is 1 1 1 1 1 1 1 and the result 1 2 3 4 5 6 7 8 .

The method seems simple and efficient, but it can hardly be used in any but rather trivial cases (for low-order polynomials) The reasons for this limitation are described below.

Suppose we have the following situation:

There are $n+1$ entries in the input-sequence.

The polynomial in question is of k:th order.

E errors are permitted.

A typical difference-table may be represented as:

```
  .    .    .    .   X    .    .    .    .
     .    .    .   X   X    .    .    .
        .    .   X    X    X    .    .
           .   X    X    X    X    .
```

X = irregular entry

. = regular entry

The number of entries at the Δ^k level is $n + 1 - k$.

At level Δ^k, each original error can affect $k + 1$ positions.

We, therefore, find that a majority of "constant" entries at level Δ^k requires that the original sequence is of length $n + 1$, where:

$$n + 1 \geq 2E(k + 1) + k$$ [16]

The sequence-extrapolator of Section 6.2 requires $k + 1$ consecutive correct entries in order to extrapolate a polynomial of order k. The situation can be illustrated as:[17]

$$\underbrace{.\quad .\quad .\quad .}_{k}\ X\ \underbrace{.\quad .\quad .\quad .}_{k}\ X\ \underbrace{.\quad .\quad .\quad .}_{k+1}$$

In this case, the required number of entries is expressed by:

$$n + 1 \geq k(E + 1) + 1 .$$

A still more efficient method is provided by Newton's general interpolation-formula, which would require:

$$n + 1 \geq k + 1 + E .$$

The difference in efficiency between the first two approaches can be written as:

$$\begin{aligned} \text{Diff.} &= 2E(k + 1) + k - k(E + 1) + 1 \\ &= E(k + 2) - 1 . \end{aligned}$$

The difference is thus dominated by the product $E \cdot k$, which indicates that the first method is unsuitable for higher order polynomials.

FOOTNOTES AND REFERENCES FOR SECTION 4

1. This formulation is due to Thomas A. Cowan.

2. Compare Section 6.

3. One extra-polator belonging to this class is described by Malcolm Pivar and Mark Finkelstein in "Automation, Using LISP, of Inductive on Sequences," in "The Programming-Language LISP," edited by Berkeley and Bobrow, Cambridge, Massachusetts, 1964.

4. An underlined element of a sequence represents a prediction or correction made by a sequence-extrapolator.

5. E2's representations of examples 4, 5, and 6 are described in more detail in Section 6.3.2.

6. See [3].

7. See "Human Acquisition of Concepts for Sequential Patterns" by H. A. Simon and K. Kotovsky in Psychological Review, 1963, Vol. 70, No. 6, p. 534-546.

8. See [7].

9. See [3].

10. See [7], p. 536.

11. The necessary capabilities are easily implemented. See section 7.

12. Actually list processing languages such as IPL-V, and LISP favor such recursion.

13. In a perverse case, $(m-1) \cdot 6 + \frac{n}{m} - 1$ tests may be required.

14. See note [3].

15. Some concepts discussed in Section 6 are employed here.

16. The expression is derived from $E(k + 1) \leq \frac{1}{2} (n + 1 - k)$.

17. The illustrated spacing of errors represents the worst case. Only at one place is it possible to find k+1 consecutive error-free entries.

5. On the Executive Function

The tasks of executives for inquiring-systems, and particularly so in non-minimum a priori designs are strongly context-dependent (compare section 2.3). The organization of executives, on the other hand, can often be designed independently of context. In the present section we will therefore employ a particular complex sequence-extrapolation inquirer SEP[1] to exemplify an essentially generally applicable executive design.

5.1 Organization of a Complex Sequence Extrapolator

It has previously, in Section 4, been shown that sequence-extrapolation, as the general representational problem, is strongly context-dependent and that, for this reason, no single extrapolator can be expected to operate efficiently over a wide domain. However, by combining several extrapolators into one complex machine, an inquiring-system with, as well, wide domain as efficient operation can be designed.

Employment of specialized sequence-extrapolators, or in more general, Kantian terms maximal _a priori_ sciences, means that the problem-solving is likely to be biased, such that problems are, non-selectively, squeezed into available models, and furthermore there is a possibility that the chosen domain of the inquirer is too narrow to permit interesting problem-solving. The problem of widening the domain is easily solved as an extrapolator corresponding to the Kantian minimal _a priori_ may be added, thus permitting a gradual model-building whenever more specialized _a prioris_ fail. Such organization permits efficient inquiry over well known domains as well as a great degree of flexibility over more general areas. Though the risk for bias of course still remains, by completely eliminating it no contextual information can be advantageously employed. Therefore, a judicious balance will have to be established between efficiency and bias, such that optimal performance is achieved.

As shown in Section 3, the available sequence-extrapolators are means for utilizing alternate representations of sequences. Consequently, the task of the complex extrapolator consists of two phases:

1. the choice of an efficient representation
2. the application of appropriate methods of extrapolation

Phase 1 will be described in the present section, phase 2 is in Sections 6 and 7.

5.2 The Domain of the Complex Sequence Extrapolator

Before proceeding some notation will have to be introduced.

M^k = a sequence-extrapolator, $k = 1,2,3,4,5$

X^k = the problem-domain of M^k

M = a set of sequence-extrapolators M^k operating under a common executive.

X = the problem-domain of M

X_i = a particular problem, $X_i \in X$, $i=1,2,3, \ldots$

Y_i = the solution to problem X_i

The application of a sequence-extrapolator, or hereafter "machine," to a problem is denoted $M^k(X_i)$. In this case M^k represents a function and X_i is its argument. The value of $M^k(X_i)$ is Y_i if a solution is found, otherwise it is undefined.

The presently implemented machines are:

M^1 : Polynomial machine

M^2 : Extended polynomial machine

M^3 : Extrapolator for intertwined sequences

M^4 : Recognizer and retrieval machine

and M^5 : Complex machine for letter-sequences.

The problem-domains of the machines are related as follows:

$X^1 \cap X^2$ = Non-empty (approximately X^1)

$X^1 \cap X^3$ = Non-empty (approximately X^1)

$X^1 \cap X^4$ = Non-empty (very small)

$X^1 \cap X^5$ = Non-empty (very small)

$X^2 \cap X^3$ = Non-empty (approximately X^2)

$X^2 \cap X^4$ = Non-empty (small)

$X^2 \cap X^5$ = Non-empty (very small)

$X^3 \cap X^4$ = Non-empty (small)

$X^3 \cap X^5$ = Non-empty (very small)

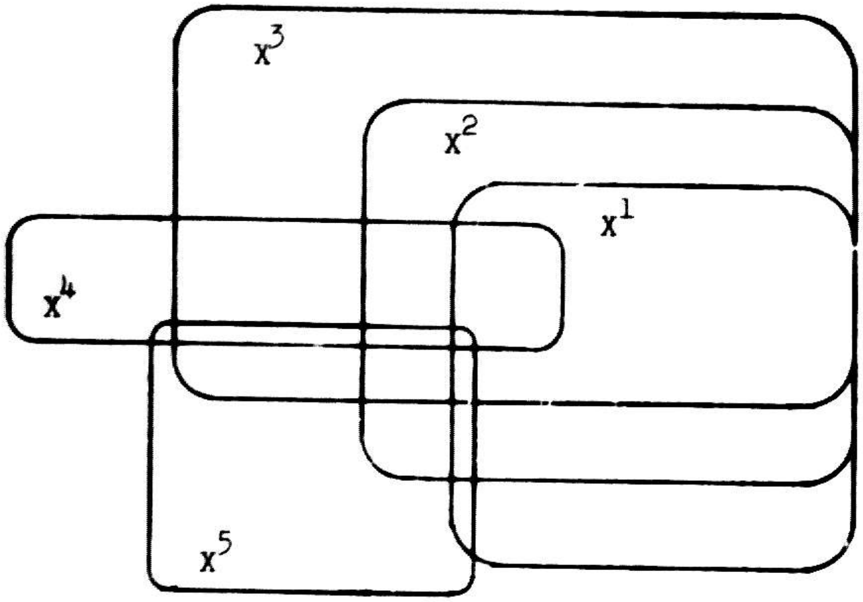
$X^4 \cap X^5$ = Non-empty (very small)

The complex machine M has a domain X which represents the union of the domains X^k of the machines M^k.

$$X = X^1 \cup X^2 \cup X^3 \cup X^4 \cup X^5$$

i.e., $X = \bigcup_{k=1}^{5} X^k$. The above relations are illustrated in figure 5.1.

Figure 5.1



Any $X_i \in X$ can by the definition of X be successfully solved by some machine M^k. If $X_i \in X^k$ it is said to be of the problem-type X^k. The type of a problem can not be determined *a priori*, only when a solution is found can its type be established. i.e.
$X_i \in X^k$ iff $V\{M^k(X_i)\} = 1$, where the function $V\{A\} = 1$ iff its argument A defines a proper solution.

The executive functions of the complex machine M may employ a great variety of strategies, ranging from simple sequential application of sub-machines to elaborate predictive strategies which sweep in all conceivable aspects of the inquiring-process. Regardless of the complexity, however, the goal of the executive is to propose optimal representations, i.e., to allocate the work between its sub-machines such taht:

1. A solution is found for any problem $X_i \in X$.
2. The average amount of information required to find a solution is minimized

3. The expected amount of time (computational effort) necessary for the identification of a solution is minimized.

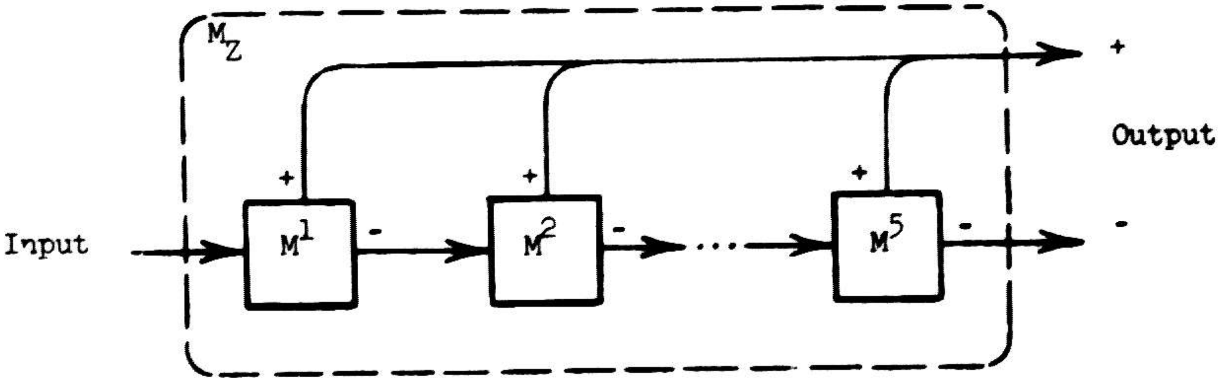
The first goal causes no difficulty in the design of the executive machine, but the other two are in a sense contradictory and do, therefore, require rather complicated analyses for a high degree of simultaneous achievement.

The next few sections will illustrate a gradually implemented complex executive for a set of extrapolators.

5.3 The Executive Functions of a Closed Machine

Hereafter a purely Leibnitzian inquiring-system is called a closed machine. Such machine may be simple, e.g. M^k, or it may be complex e.g., M. A closed complex machine M_S is defined by any predetermined sequence Z of application of simple machines M^k. Figure 5.2 illustrates a simple case.

Flow of Control in a Closed Machine



+ = success
- = failure

Figure 5.2

A closed complex machine M_Z is defined such that its sub-machines are employed in the order: $M^{k_1}, M^{k_2}, M^{k_3}, M^{k_4}, M^{k_5}$ where the sequence of superscripts stands for any predetermined permutation of machines, $k_i \epsilon (1,2,3,4,5)$ and $i=1,2,3,4,5$.

Machine M_Z satisfies goal one because successive application of each of the five sub-machines covers the problem-domain X. Thus, any $X_i \epsilon X$ is solvable provided that a sufficient amount of information about X_i is available.

M_Z does not, in general, achieve goals two and/or three, a deficiency which may be partly removed by an appropriate organization of the closed machine.

Goal 2 may be realized by the following procedure:

1 Introduce the smallest possible amount of information about X_i into the machine M_Z. Denote this amount of information by $1q(X_i)$; (in general, an amount of j quanta of information about X_i is denoted by $jq(X_i)$, i.e., here $j = 1$).

2 Apply M_Z to $jq(X_i)$. If a solution is found goals 1 and 2 are achieved.

3. If no solution is found in step 2, introduce a new quantum of information, i.e., substitute $(j+1)q$ for jq. Then return to step 2.

By applying the machine M_Z to increasingly large amounts of information until a solution is found, we know that within the size of one quantum a minimum amount of information is employed.

In symbolic form, the procedure may be written as:

Find the lowest value $\bar{j}$ of j for which

$$V\{M_Z(\bar{j}q(X_i))\} = 1 \tag{1}$$

Where $V\{A\} = 1$ if and only if its argument A defines a solution.

In general, this procedure will defeat the realization of goal three and particularly so when information is added in very small quanta. If we assume that the time requirement for a solution is independent of the amount of information employed, then goal three may be achieved by introducing any amount of information larger than $\bar{j}q(X_i)$. As we do not have any a priori information about the size of $\bar{j}$, a time-minimizing strategy would have to apply all information available about X_i in order to assure successful solution of X_i in the first attempt.

Symbolically this strategy may be written as;

$$V\{M_Z(\max q(X_i))\} \equiv 1$$

Where $\max q(X_i)$ = all available information about X_i.[3]

Goals two and three are simultaneously satisfied only when $\bar{j}q(X_i) = \max q(X_i)$. Depending upon the relative importance attached to the two goals, some compromise such as introduction of an amount mq of information, $\bar{j}q(X_i) < mq(X_i) < \max q(X_i)$, may be justified.[4]

Further improvements of the efficiency of M are not possible without relaxing the assumption of a closed machine M_Z.

5.4 The Executive Functions of an Adaptive Machine

By relaxing the restriction of using a closed machine, information about previous performance can be used to improve the efficiency of the executive.

An easily implemented improvement consists in assigning thresholds $\bar{q}^k$ to each sub-machine M^k such that it does not accept any amount of information below $\bar{q}^k$, i.e., $jq(X_i) \geq \bar{q}^k$ is required. A machine employing such thresholds is illustrated in Figure 5.3

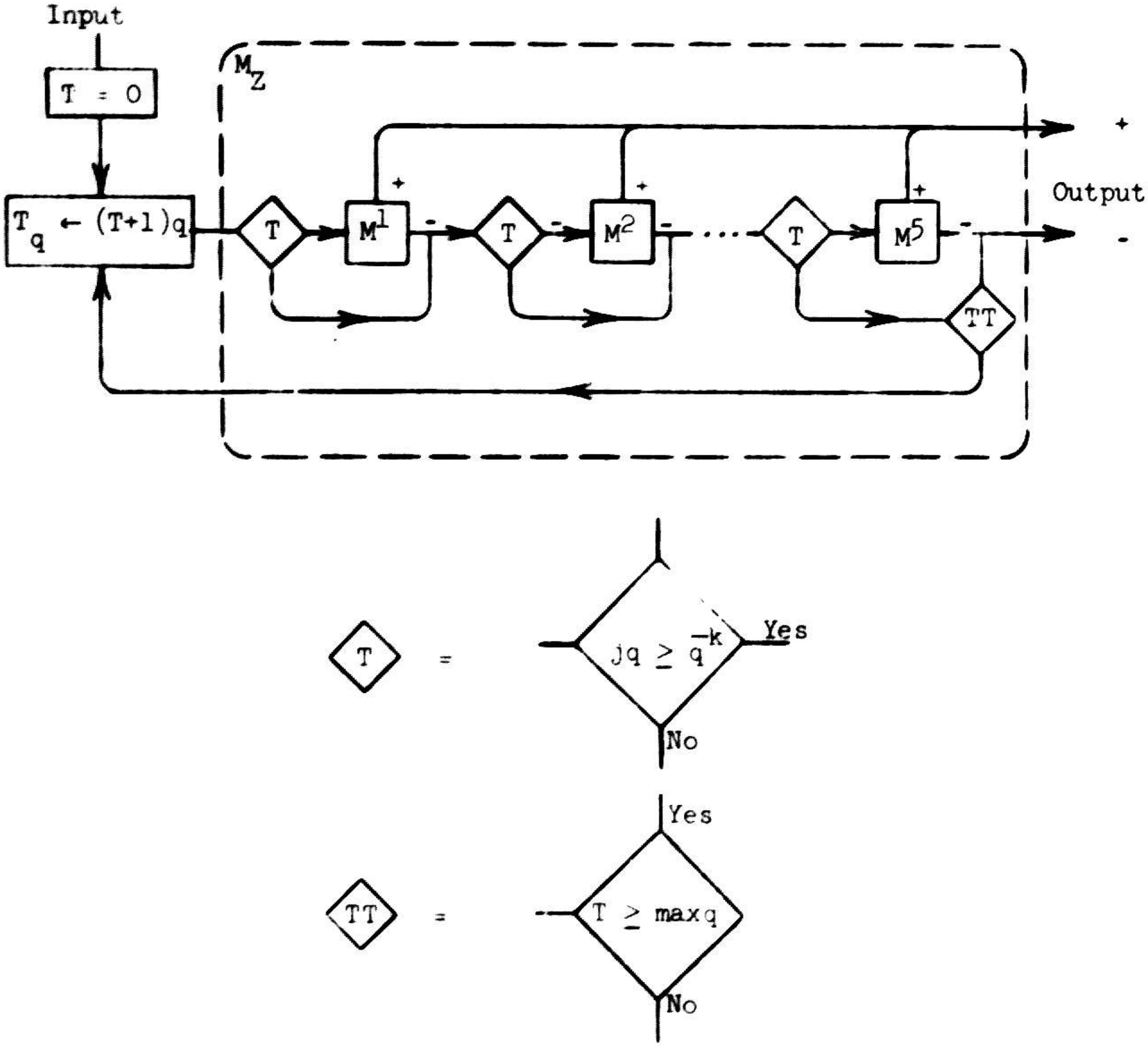


Figure 5.3

$\bar{q}^k$ can be determined as

a. the smallest amount of information that has ever sufficed to solve a problem $X_i \epsilon X^k$; or

b. the expected amount of information necessary to solve a problem $X_i \epsilon X^k$.

In both cases $\bar{q}^k$ can be determined from information about previously solved problems.

In symbolic nota+ion we have:

a.
$$\bar{q}^k = \min_n j_n q(X_i^k) \tag{3}$$

or b.
$$\bar{q}^k = E(jq(X_i)) = \frac{1}{r^k} \cdot \sum_{n=1}^{r^k} j_n q(X_i^k) \tag{4}$$

$n=1,2 \ldots, r^k$

$k=1,2,3,4,5$

r^k = The number of problems of type X^k encountered.

$j_n q(X_i^k)$ = The amount of information about the n:th problem of type X^k which was found necessary for its solution.

A simple example may clarify the use of the thresholds $\bar{q}^k$:

Suppose the sequence of sub-machines defining M_Z is:

$$\langle M^1, M^2, M^3, M^4, M^5 \rangle$$

Assume the following thresholds have been derived from previous experience:

Sub-machine	Threshold
M^1	$\bar{q}^1 = 4$
M^2	$\bar{q}^2 = 5$
M^3	$\bar{q}^3 = 6$
M^4	$\bar{q}^4 = 3$
M^5	$\bar{q}^5 = 4$

In the process of solving a problem X_i, the sub-machines would be applied in the following order:

$(3q(X_i))$: M^4

$(4q(X_i))$: M^1, M^4, M^5

$(5q(X_i))$: M^1, M^2, M^4, M^5

$(6q(X_i))$: M^1, M^2, M^3, M^4, M^5

$(7q(X_i))$: M^1, M^2, M^3, M^4, M^5

etc.

The sequence is terminated when a solution is found.

Further adaptation may be made in response to information gained about the relative frequencies ψ^k of occurrence of the different problem-types X^k $(k=1,2,3,4,5)$. If available such information may be used to minimize the expected time-requirement for finding a solution.

The minimization may be expressed as:

Choose the permutation Z of sub-machines for which the expected time for solution of a problem X_i is minimized, i.e., find:

$$\operatorname*{Min}_{Z} \sum_{k=1}^{5} \psi^k \tau_Z^k$$

$$\tau_Z^k = \sum_{k_j=1}^{k} \left(\bar{\varphi}_{k_j-1,k} \cdot \varphi_{k_j,k} \cdot \sum_{i=1}^{k_j} T^{M^{k_j}}\right) \qquad (5)$$

Where M_S = A given permutation of machines M^{k_j} where

$k_j \epsilon \{1,2,3,4,5\}$.

τ_Z^k = Expected time for finding a solution to a problem of type X^k on the machine M_Z

$(S = 1,2,\ldots, 120)$.

$\bar{\varphi}k_j,k$ = The probability that the machine M^{k_j} will solve a problem of type X^k.

$T^{M^{k_j}}$ = The expected time for solution (or indication of failure) of a problem X_i on machine M^{k_j} .

A simple example may indicate how τ_S^k is determined:

Suppose: $M_Z = \langle M^2, M^4, M^3, M^1, M^5 \rangle$

$\varphi 2,4 = 0.25$

$\varphi 4.4 = 1.00$

$\tau_Z^4 = 0.25T^{M^2} + (1 - 0.25)\,(T^{M^2} + T^{M^4})$

The procedures for increasing the efficiency of the executive function of a problem solver M, as outlined here, may of course also be based on information received from external sources. Unless such information can be received and acted upon continuously the potential for improvement is limited to a given environment, i.e., we still have a basically closed machine. Such machine can, however, itself be used to collect information about the problem types, to compute its own "optimal" organization, and to revise said organization if the environment changes. This gives us a fully adaptive machine which will be "optimal" in some sense over a wide range of different environments.

The task of the executive is to optimize the permutation of submachines at time t, denoted $Z(t)$ on the basis of statistics of performance. Any one of a number of adaptive procedures E may be employed.

$$Z(t) = E[Z(t-1)] \tag{6}$$

Improvements in efficiency due to the executive function has, until now, been limited to rather simple rescheduling of the flow of control between sub-machines. However, given well-defined problems, i.e., a guarantor for the correctness of produced extrapolations, an introspective executive (compare Lockean inquiring systems) will be able to perform a more sophisticated allocation of work. The next section will illustrate such executive.

5.5 Inquiring Executives

We have previously indicated that it would be necessary to know the type (class-belonging X^k) of the problem X_i for simultaneous achievement of goals two and three. This information is not directly derivable from the formulation of the problem X_i, but it may still be possible to predict the class-belonging of any given X_i. We have been concerned with statistical measures for improving the expected performance of the basic machine. Now we intend to investigate methods to predict the problem-type for specific problems, i.e., we want to find some reasonably efficient way of predicting the next entry in the time-sequence of problem-types encountered by the machine.

To do reasonable predictions the executive needs a model of its environment and our task will now be to explore the possibilities of generating such a model.

A useful model of the environment should be formulated in terms of the processes available within the system, and furthermore, it should also be manipulated and interpreted internally. These restrictions, of course, limit the possibilities but it is still feasible to design a useful model provided certain a priori assumptions are shown to be justified.

The task of the model should be to permit prediction of the type of any given sequence in order to assign suitable extraplators. Such prediction can only, to a very limited extent, be made by studying the properties of input-sequences; however, by assuming such regularity in the environment that input-sequences, if observed in their context would be parts of a pattern, then prediction would be possible if only the pattern could be revealed. The only context, within which the executive can search for information, is the sequence in which different problem-types have been received. Therefore, the following assumptions will have to be made:

1. that there exists some pattern or strategy according to which problems are generated and presented, and

2. that, given such pattern, its categories are homomorph to the problem-types which can be recognized by the inquirer M .

Assumptions 1 and 2 imply that the context of events (problem-types) can be represented as a sequence of problem-types. Prediction can then be performed provided that some suitable sequence-extrapolator is available.

The situation is slightly re-formulated in the following description:

1. the machine M receives its problems X_i according to some strategy S for presentation of problem-types X^k, and

2. the strategy S is operational for the executive E, i.e., E can extrapolate a sequence of problem-types generated according to S .

Figure 5.4 illustrates the functions of the executive.

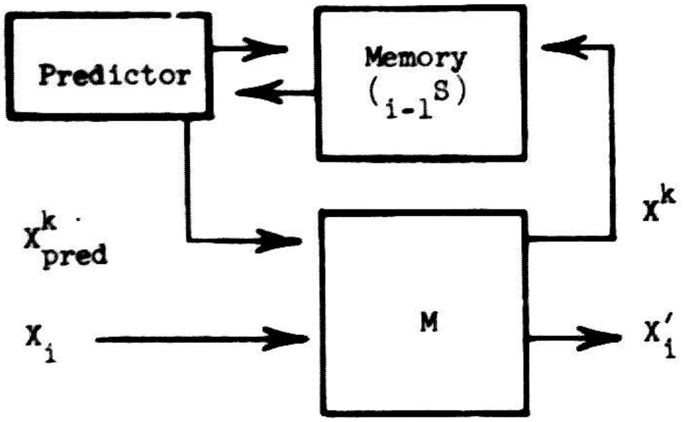


Figure 5.4

X^i	= input-sequence
X_i'	= extrapolated X_i
X^k	= problem-type
X^k_{pred}	= predicted problem-type
${}_{i-1}S$	= stored parts of the input-strategy
M	= complex extrapolator

In Figure 5.4 the executive receives problems X_i in a sequence determined by a previously chosen strategy. The executive E, on arrival of a problem, retrieves a sequence, denoted ${}_{i-1}S$, of the i-1 latest encountered problem-types. The sequence ${}_{i-1}S$ corresponds to the first i-1 positions of the strategy S . The executive, by employing a predictor, i.e., sequence-extrapolator, attempts to predict the next problem. Assuming that this is most likely to be of type X^k, machine M^k is chosen. Whenever a solution is found, the name of the successful submachine is stored in the memory as the type of the corresponding problem.

The procedure for problem-solving is as follows:

1. The class-belonging X^k of X_i is predicted by E,

   i.e., $$E({}_{i-1}S) \rightarrow X_i^k \quad (\text{i.e., } X_i \epsilon X^k) \quad .$$

2. Assuming the prediction of the class-belonging to be correct, the sub-machine M^k is applied to the problem X_i,

   i.e., $$M^k(X_i) \rightarrow Y_i \quad .$$

3. If a solution is found, the "memory" is updated,

   i.e., $${}_{i-1}S + [M^k] \rightarrow {}_iS \quad ,$$

otherwise, some other class-belonging is assumed and points 2-3 are repeated.

Obviously the prediction of the problem-type X^k for a given X_i is a problem which may be attacked in exactly the same way as solving the problem X_i, itself. It is, therefore, possible to extend the complexity of the executive E as far as we have been able to extend the organization of the problem-solving machine M, itself.

5.6 Complex Inquiring Executive

It is quite possible that there may be several alternate types of strategies of presenting problems to the inquirer. If this is the case, the question of efficient domain for the executive will arise. Thus, the executive, too, may be most efficiently organized as a complex machine. In thise case, the external problem-poser can be assumed to employ a strategy of presenting types of strategies for presenting problems to the inquirer M . As seen, the situation at the strategy-level parallels that of the problem-level.

For convenience, we may think of each available strategy-type as an 'experimenter' who has a particular way of presenting problems. The situation resembles the frequent case where students learn about their professor's strategy of formulating test-problems in order to optimize their problem-solving performance, but as every new professor employs his own strategy, they have to be alert to changes.

In summary, the situation is:

1. the machine M receives its problems X_i from any one of several experimenters;

2. each experimenter employs his own stragegy of presentation of problems;

3. the executive has models for prediction of the probable strategies, but is not informed which model is applicable at any given time.

An important goal of the executive is to identify the experimenter, or rather the current strategy, before trying to solve any given problem X_i .

As there is no a priori knowledge available about the identity of the experimentor, we have to find decision-rules for identification of efficient machines. There are several possible rules, some of which are listed below:

1. Committee-vote: Let each sub-machine of the executive make a prediction, choose the vote of the majority. This approach could be used in a two-choice situation, for multiple choices it might bring confusion. Furthermore, as only one prediction can be correct, no compromise solution is likely to be satisfactory.
2. Priority-choice: (Dictatorship) In this case one sub-machine of the executive is given priority, such that when it is able to produce a prediction, this will be chosen. This method may, in certain instances, be very powerful. The design of the priority-scheme, however, requires certain a priori information about the environment.
3. Competition: Let all machines provide predictions and pick the "opinion" of the machine which in most cases has proven to be correct. This may require very extensive double-computation unless a computationally efficient approach is chosen. Some approaches are listed below:
   - a. In stage 1, experience is collected and the prediction of each sub-machine is used to find a suitable representation of X_i . A corresponding method of problem-solving is applied. The amount of time used for each of the methods of solution is recorded. In stage 2, experience is employed to choose the sub-machine which has the lowest expected time-requirement.

b. The prediction of a randomly chosen sub-machine is used to choose representation. The probability of picking a particular machine is determined by statistics of past performance, which are updated as soon as success or failure of a prediction is established.

c. All sub-machines make predictions. One of these is chosen by random for solution of the problem. The time requirement is recorded. Thereafter, knowing the actual problem-type, each of the remaining predictions is employed as a starting-point for simulated problem-solving. The expected time-requirements are computed and compared, then the representation corresponding to the lowest time-requirement is chosen. (Section 5.6.1 gives a more detailed description of this approach.)

5.6.1 Employment of a Model of the Complex Machine

The situation facing the complex problem-solver of Section 5.6 may be briefly stated as:

There is a community of inquirers which is to decide about the categorization of a particular element, which cannot be analyzed by direct observation, but which may be indirectly approachable via its membership in a sequence of presentations. Such membership can be, as shown, established by brute-force methods, though, simpler and more efficient methods are to be preferred. A detailed account of one such method, namely employment of a simulated model of the executive, will now be given. The method is most conveniently described as a sequence of steps:

1. Apply a probability vector to pick one sub-machine by random. Solve the problem X_i as suggested by the prediction. Record the computational time used at each step of the problem-solving process and store the results at each particular sub-machine. (When a sub-machine has been previously "timed", a stored average is updated.) Also, keep track of the real cumulative time used up to and including each stage of the solution process.

2. When a solution is found, the type X^k of the corresponding problem X_i is known.

3. For each sub-machine not used in step 1., simulate the problem-solving which would have been necessary for finding a solution, starting at the point defined by the prediction of step 1. (A detailed description of this stage will be given below.)

4. Compare the time used for actual problem-solving with those time-requirements which have been derived by simulated problem-solving. Increase the probability of picking the machine which provided the smallest time-requirement (regardless of whether the time-requirement was derived by actual or simulated problem-solving). This ensures that the machine which has been most successful in predicting the correct problem-types will be most likely to provide the prediction used for future problem-solving.

By previous assumptions about the nature of the problem-domain, we know that any sequence of application of all sub-machines M^k will, providing that sufficient information about X_i is given, ultimately find a solution. The permutation of sub-machines defined by the structure of M_Z, augmented by thresholds for minimum and maximum amounts of information used by the different sub-machines defines a path between any starting-point and a solution. An example is given in Tables 5.1 and 5.2.

EXAMPLE:

A problem X_i is received and steps 1 and 2 of the above sequence have been performed.

Suppose that the solution found in step 1, and recognized in step 2, indicates problem-type 3 requiring $7q$ of information.

Suppose that some other sub-machine G predicts the problem to be of type 2, and that past experience indicates that problem-type 2 requires at least $6q$ of information.

The two assumptions give a starting-point type 2, $6q$ and an end-point type 3, $7q$ for the simulated problem-solving of a machine defined in Table 5.1. The situation is illustrated in Table 5.2. The expected time for solution of the problem is computed to be T.

Parameters of the machine M_Z

$$M_Z = \langle M^1, M^2, M^3, M^4, M^5 \rangle$$

Machine number	Minimum threshold	Maximum threshold
1	3q	6q
2	4q	8q
3	6q	10q
4	4q	4q
5	8q	10q

Table 5.1

Simulated problem-solving by M_Z

	Sequence of application of sub-machines	Information quantity	Average time required in previous use	Cumulated time for the current problem
	1	3q	t_{13}	
	1	4q	t_{14}	
	2	4q	t_{24}	
	4	4q	t_{44}	
	1	5q	t_{15}	
	2	5q	t_{25}	
	1	6q	t_{16}	
Start →	2	6q	t_{26}	t_{26}
(predicted)	3	6q	t_{36}	$t_{26} + t_{36}$
	2	7q	t_{27}	$t_{26} + t_{36} + t_{27}$
Solution →	3	7q	t_{37}	$T = t_{26} + t_{36} + t_{27} + t_{37}$
(achieved)				

Table 5.2

5.6.1.1 A Note on Simulated Problem-solving

It may turn out to be a rather difficult task to construct a new table for every encountered problem (N·B! thresholds and flow of control may vary according to experience.) Therefore, the machine itself is employed as a generalized table, i.e., the machine is used to describe itself. The method is rather straightforward. A simple illustration is given in Figure 5.5.

The mode of operation of the sub-machine M^k is determined by the executive machine. During the actual problem-solving phase, the computation mode is chosen, i.e., the necessary computations are performed and the amount of time $t_{k,i}$[5] used by M^k is stored in a location accessible during the simulated mode.[6] In the simulated mode, the time $t_{k,i}$ is picked up after which the actual computational part of M^k is by-passed. Control is thereafter transferred to the failure-exit or the success-exit, depending upon the relation between the type of the problem X_i (as decided by actual problem-solving) and the domain of the machine M^k under test.

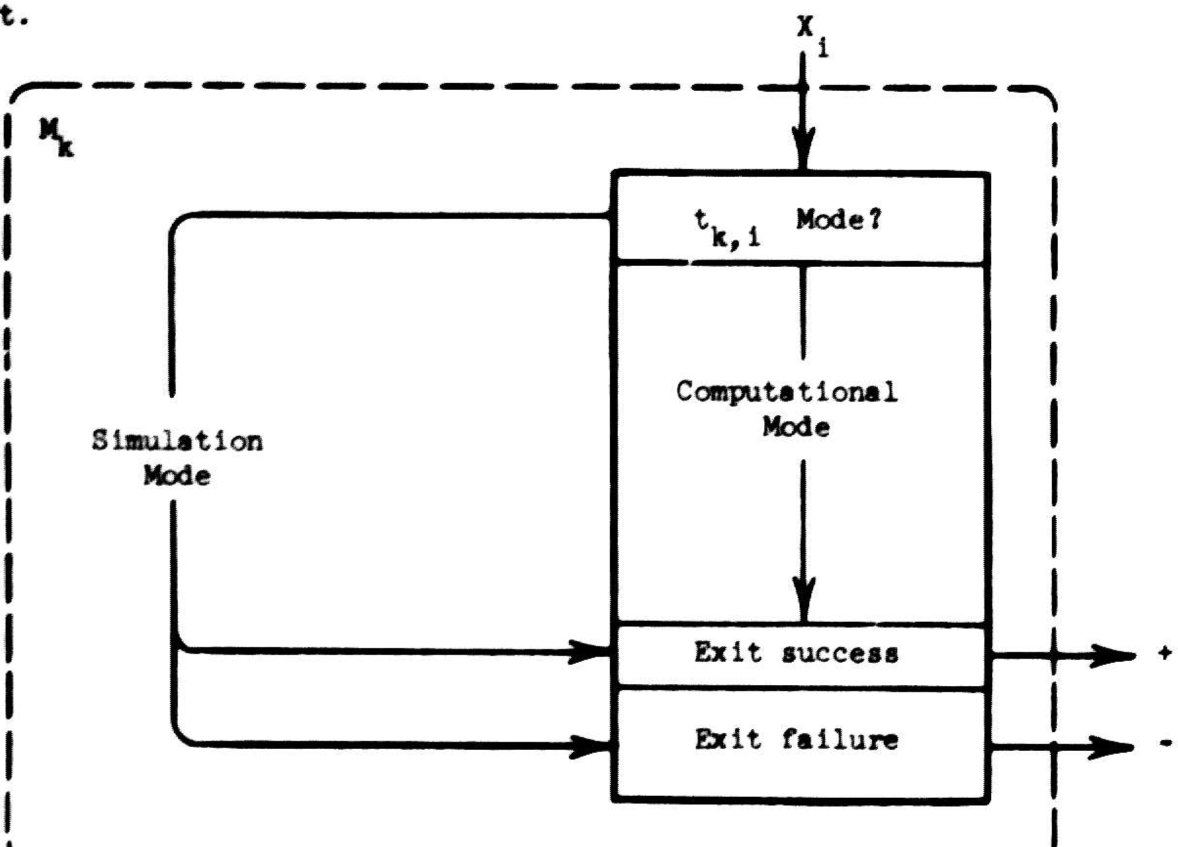


Figure 5.5

By providing each sub-machine with a facility for by-passing most of its time-consuming parts, we have, in effect, facilitated introduction of partially decentralized control which permits the use of a very simple executive machine. When the sub-machines are made "responsible" for performing most tests for feasibility, constraints, etc., the executive machine essentially transfers control between the sub-machines in a round-robin-fashion (with provisions made for high level constraints). The sub-machines thus can be used to make a majority of decisions concerning their own relevance for a given situation.

Some advantages of this approach are:

1. Any number of sub-machines (with very differing applicability) may be easily connected to the executive machine without greatly increasing the total level of complexity.

2. Any particular constraint may be included into a sub-machine without necessitating any changes in the executive organization.

3. The feature of "simulating" the time-requirement for using the sub-machines may be easily implemented.

5.7 Exploration of a "Super-strategy"

The situation, as previously indicated, may well be that problems are presented according to alternate strategies which, for instance, could be the case when experimenters are alternating according to some "super-strategy". By trying to identify such super-strategy, the executive implicitly assumes that there are multiple levels of causation of events. One possibility of doing this will be outlined.

To describe the super-strategy SS, we need an indicator which at any given point of time can be used to find the class-belonging of the current strategy. As we have previously shown, the simulation approach may be used to identify the best predictor for a sequence of problem-types X^k . We have also indicated that the probability of applying a particular predictor is positively correlated to its current degree of success, i.e., after a large number of successful predictions by a machine, the probability of applying it will be greater than that of any other machine. But if the current strategy is replaced by some other (in accordance with the super-strategy SS), then the probability-vector for choosing predictor will adapt to the new situation. Changes in said probability-vector thus indicate the sequence of strategy-types applied by SS . We thus can conclude that variations in the probability-vector for choosing prediction machines indicates changes initiated by the super-strategy SS .

The situation is illustrated in Figure 5.6.

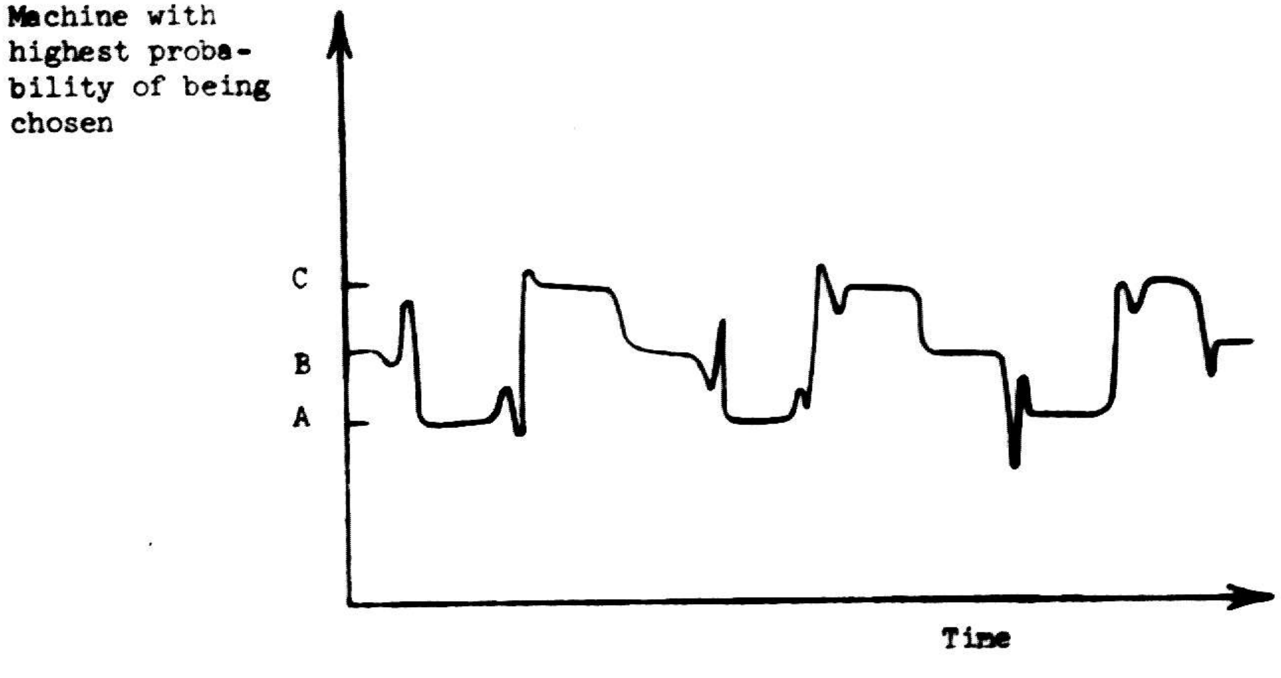


Figure 5.6

Disregarding transitory oscillations, we, from Figure 5.6, find that the sub-machines, i.e., the classes of strategies, have been applied in the following sequence:

B A C B A C B A C

The executive is a generally applicable machine for identification of strategies and can thus be used to find the super-strategy SS by extrapolating the above sequence.

Two levels of prediction have been described. One may be identified as prediction of problem-types under a non-changing environment, the other predicts changes in the environment itself. At any point of time the strategy in use is assumed to be known, thus permitting a successful prediction of the sequence of future problem-types. When the strategy is changed, this is manifested as an oscillation in the previously discussed probability-vector. Any change in said vector triggers a rapid adaptation to a predicted change in the environment.

Footnotes for Section 5.

1. SEP (Sequence Extrapolator) is a program for a digital computer written in the programming-language IPL-V.

2. There are 120 such permutations.

3. Any $X_i \epsilon X$ will be solved in the first application of M_Z .

4. The cost of information is rarely discussed in the literature on decision-making. A notable exception, however, is Marshak and Radners' work on the theory of teams.

5. In most instances $t_{k,i}$ is computed as an average time-requirement for several solved problems of type k .

6. A "timing-routine", in the form of an operation-cycle counter, can be inserted at any part of an IPL-V program during the execution-phase.

6. Extrapolators for Sequences of Numbers

Representation and modeling in the general area of inquiry as well as in sequence extrapolation has in preceding sections been discussed with only superficial connection to particular programs for digital computers. In this and the following section a description will be given of several implemented sequence extrapolating programs. The discussion will be given at a sufficiently detailed level to permit reconstruction of the basic ideas of the programs, however, particular aspects of programming will not be discussed.

The sequence-extrapolators presented are all "complete" in the sense that they can be employed as isolated Leibnitzian systems and, furthermore, most of them are also homogeneous such that their input-output specifications permit connection to SEP. A program can be included in SEP by simply adding its name to a list of members. That is, all transfers of information between SEP and its members are performed via "public" communication cells, and thus great flexibility in organization is achieved.

6.1 Extrapolation of Polynomial Sequences

The following basic scheme will correctly extrapolate any sequence of m numbers generated by a polynomial of a degree lower than m.

A polynomial sequence $Y = y_o, y_1, \ldots, y_n$ is defined as a tabulation of a polynomial $f(x)$ for the values

$$y_s = f(x_s) \qquad s = 0,1,2,\ldots, n$$

$$x_s = x_o + s \cdot h \qquad h = 1$$

$$x_o = 0$$

The k:th forward difference of x_s denoted $\Delta^k x_s$ is defined by:

$$\Delta^k f(x_s) = \Delta^{k-1} f(x_s + h) - \Delta^{k-1} f(x_s)$$

$$\Delta^0 f(x_s) = f(x_s) \ .$$

Rearranging terms

$$\Delta^{k-1} f(x_s + h) = \Delta^{k-1} f(x_s) + \Delta^k f(x_s) \qquad (1)$$

Denote the set of polynomials of order m by P^m. For any $f(x) \epsilon P^m$ (2)

$$\Delta f(x) \epsilon P^{m-1}$$

$$\Delta^m f(x) \epsilon P^0 \quad \text{or} \quad \Delta^m f(x_s) = \Delta^m f(x_{s+1}) \qquad (3)$$

$$\Delta^{m+j} f(x_s) = 0 \qquad j=1,2,\ldots \qquad (4)$$

By (1) and (3), any polynomial sequence $Y = y_o, y_1, \ldots, y_n$ $(n \geq m)$ can be extrapolated by the following scheme:

$$\Delta^m f(n - (m - 1)) = \Delta^m f(n - m)$$

$$\Delta^{m-1} f(n - (m - 2)) = \Delta^{m-1} f(n - (m-1)) + \Delta^m f(n - (m-1))$$

$$\vdots$$

$$\Delta f(n) = \Delta f(n - 1) + \Delta^2 f(n - 1)$$

$$f(n + 1) = f(n) + \Delta f(n)$$

or

$$y_{n+1} = y_n + \Delta y_n$$

The procedure is most conveniently demonstrated in a difference-table (figure 6.0).

$$f(X) \; : \quad f(x_o) \;\; f(x_1) \; f(x_2) \quad . \quad . \quad . \quad f(x_n) \quad f(x_{n+1}) = f(x_n) + \Delta f(x_n)$$

$$\Delta f(X) \; : \quad \Delta f(x_o) \, \Delta f(x_1) \quad . \quad . \quad . \quad \Delta f(x_{n-1}) \, \Delta f(x_n) = \Delta f(x_{n-1}) + \Delta^2 f(x_{n-1})$$

$$\Delta^2 f(X) \; : \; \Delta^2 f(x_o) \quad . \quad . \quad . \quad \Delta^2 f(x_{n-2}) \Delta^2 f(x_{n-2}) + \Delta^3 f(x_{n-3})$$

.

.

.

$$\Delta^m f(X) \; : \; \Delta^m f(x_o) \qquad \Delta^m f(x_{n-m}) \Delta^m f(x_{n-(m-1)}) = \Delta^n f(x_{n-m})$$

$$\Delta^{m+1} f(X) \; : \; 0$$

Figure 6.0

Each entry of the difference-table equals the sum of its left neighbor and the entry just below said neighbor.

A simple example may illustrate the procedure. Y = 1, 4, 9, 16, 25.

$f(X)$	1	4	9	16	25	$\boxed{\underline{36} = 11 + 25}$	(2)
$\Delta f(X)$	3	5	7	9	$\boxed{\underline{11} = 9 + 2}$		
$\Delta^2 f(X)$	2	2	2	$\boxed{\underline{2} = 2}$			(1)
$\Delta^3 f(X)$	0	0					

(1) $\Delta^2 f(x_s) = \Delta^2 f(x_{s+1}) \rightarrow \Delta^2 f(x) \in P^0 \rightarrow f(x) \in P^2$

(2) $f(X) = (X + 1)^2 \qquad X = 0, 1, \ldots$

6.1.1 Error-correction in Polynomial Extrapolation

By limiting the domain of a sequence-extrapolator to polynomial sequences of degree m or less, certain error-correction-procedures based on the following theorem may be implemented.

Theorem 1: There is one and only one polynomial $f(X)$ of degree m which for $m + 1$ different arguments $x = x_i$ $(i = 0, 1, 2 \ldots m)$ assumes $m + 1$ given values $f(x_i)$.[2]

The design of the polynomial extrapolator has been based on the existence part of Theorem 1. The uniqueness part permits implementation of certain error correction facilities.

Given a polynomial input-sequence Y of a degree not exceeding m

$$Y = y_o,y_1,\dots,y_n \qquad n \geq m$$

$$Y \in \bigcup_{j=o}^{m} P^j$$

Y defines a set of argument-value pairs XxY where X = {0,1,...,n} . By Theorem 1 there is one and only one m:th degree polynomial $f : X \rightarrow Y$ such that f satisfies m+1 arbitrary values of XxY .

Sequence-extrapolation based on Newton's interpolation formula for constant intervals, however, requires m + 1 consecutive values of XxY.

Result 1: A polynomial sequence of degree m or less can be uniquely identified by any m + 1 consecutive entries.

Definition: A polynomial input-set $U = \{u_o,u_1,\dots,u_n\}$ is an arbitrary permutation of a polynomial input-sequence.

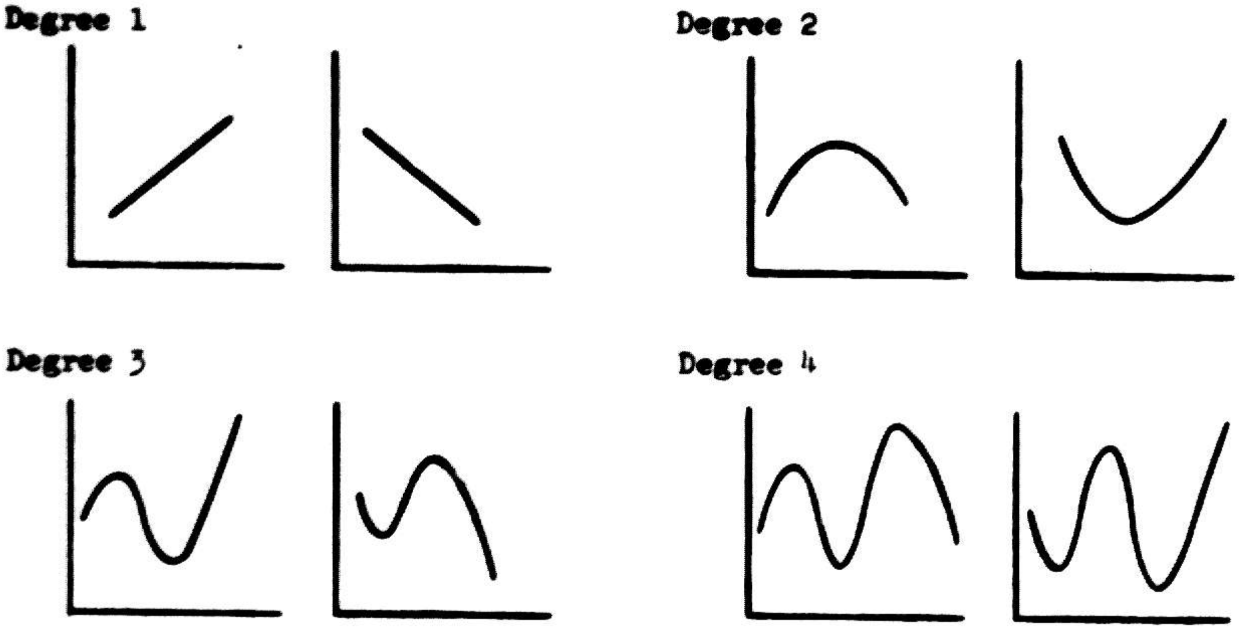
Proposition 1. Any sufficiently large polynomial input-set U can be identified, ordered, and extrapolated by the polynomial sequence-extrapolator.

In the following, a heuristic proof of Proposition 1, based on certain well known properties of polynomial functions, will be given.

Theorem 2: The derivative f'(x) of a m:th degree polynomial has at most m-1 roots.[3]

Theorem 2 insures that a polynomial sequence $f(x)$ will monotonously approach $+\infty$ or $-\infty$ for sufficiently large arguments, i.e., for arguments larger than any root of $f'(x)$.

All possible forms for graphs of polynomial sequences of degree 4 or less are given in Figure 6.1.[4]


Degree 1

Degree 2

Degree 3

Degree 4

Figure 6.1

Polynomial functions are continuous, therefore, all values corresponding to finite arguments are finite. This property in combination with Theorem 2 leads to the following result.

Result 2. There exists for any polynomial $f(x)$ a constant x^* such that

$$|f(x)| \le |f(x^*)| \quad : \quad x \le x^* \quad .$$

$$|f(x+1)| > |f(x)| \quad : \quad x > x^*$$

Based on Result 2, the following procedure will identify the polynomial input-sequence Y underlying any sufficiently large polynomial input-set U .

1. Sort the members of U into a sequence $u_1, u_2, \ldots, u_i, \ldots, u_n$

   such that

   $|u_i \leq |u_{i+1}|$ $\qquad i = 0, \ldots, n-1$

2. Find the generating polynomial g of the subsequence

   $u_{n-4}, u_{n-3}, u_{n-2}, u_{n-1}, u_n$. Provided that $n \geq x^* + 5$, Result 2 guarantees that $u_i = y_i$ for $i = n-4, \ldots, n$.

3. Generate a sequence Z where $z_i = g(i-(n-4))$ $\quad i = 0, 1, \ldots, n$

   If $n = x^* + 5$, Theorem 1 assures that $z_i = y_i$.

4. The generating polynomial $f(X)$ of Y is found by substituting $x-(n-4)$ for Z in g .

The procedure is illustrated in Figure 6.2.

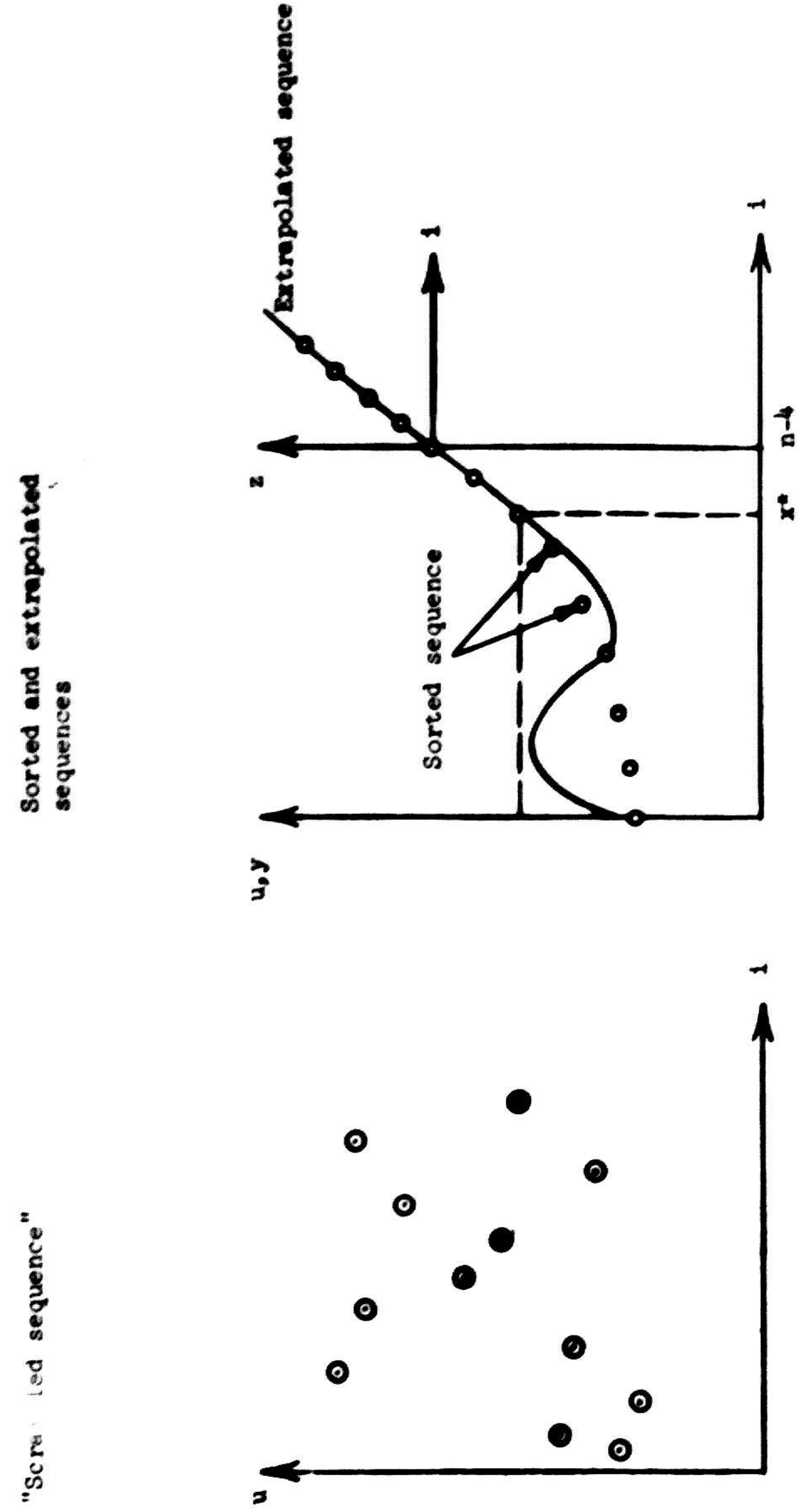

"Scrambled sequence"
u
1
Sorted and extrapolated sequences
u, y
z
Sorted sequence
Extrapolated sequence
x*
n-4
1
1

Figure 6.2

6.1.2 Computer Implementation of a Basic Polynomial Extrapolator

The procedure described in Section 6.1 has been implemented using several different programming-languages such as IPL-V, FORTRAN, PDP-5 machine-language, etc. The IPL-V version is employed by SEP for certain support-functions, however, the unrestricted generality, compare Section 4.1, makes direct application of the program unsuitable for the purposes of SEP.

A simple flow-chart of the program is given in Figure 6.3.

6.2 Identification of Generating Polynomials

The basic scheme derived from Equation 1 is applicable for extrapolation of any polynomial sequence, but it does not explicitly identify the generating polynomial $f(x)$. An adaptation of Newton's forward interpolation formula, however, permits such identification.

Newton's forward formula can be written

$$f(x_o + s \cdot h) = f(x_o) + \binom{x_o + s \cdot h}{1} \Delta f(x_o) + \binom{x_o + s \cdot h}{2} \Delta^2 f(x_o) + \dots + \binom{x_o + s \cdot h}{n} \Delta^n f(x_o) + R$$

$$= \sum_{j=o}^{n} \binom{x_o + s \cdot h}{j} \Delta^j f(x_o) + R \, . \qquad (1)$$

For the present extrapolator $x_o = 0$ and $h = 1$. Furthermore, for polynomial sequences of order m ; $R = 0 \cdot$ for $n > m$.

A Basic Polynomial Extrapolator

Input is Y^o

Produce the sequence Y^{j+1} by applying Δ to each member of Y^j.

Are all members of Y^j equal.

Add another copy of y_o^j to Y^j.

Compute and append a new entry to Y^j.

Y^o: $(y_o, y_1, \ldots, y_n, y_{n+1})$

= name of a sequence or list

(Y^j) = a sequence or list

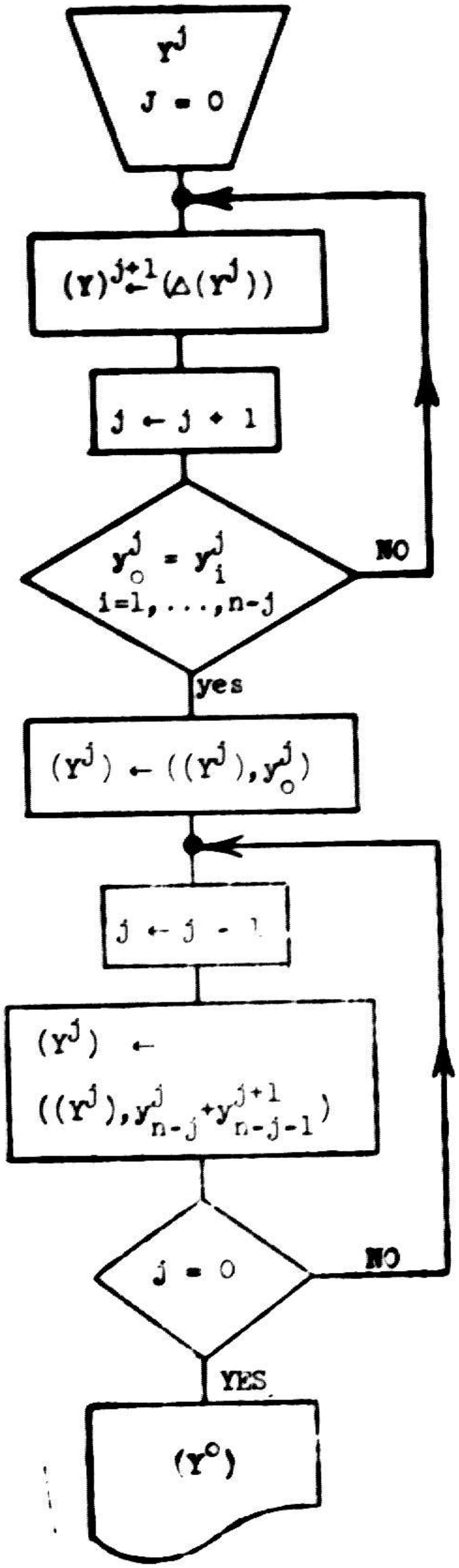


Figure 6.3

Thus:

$$f(x) = \sum_{j=o}^{n} \begin{pmatrix} s \\ j \end{pmatrix} \Delta^j f(x_o) \qquad (2)$$

Any order difference can be written as a linear combination of functional values.

$$\Delta^j f(x_o) = \sum_{k=o}^{j} (-1)^{j-k} \begin{pmatrix} j \\ k \end{pmatrix} f(x_k) \qquad (3)$$

Equations (2) and (3) give

$$f(x_s) = \sum_{j=o}^{n} \begin{pmatrix} s \\ j \end{pmatrix} \cdot \sum_{k=o}^{j} (-1)^{j-k} \begin{pmatrix} j \\ k \end{pmatrix} f(x_k) \ . \qquad (4)$$

Equation (4) is suitable for polynomial extrapolation. Substituting $n + 1$ for s gives the value of $y_{n+1} = f(x_{n+1})$.

Limiting the domain to polynomials of order $m \leq 4$, Equation (2) can be written:

$$f(x_s) = f(x_o) + s \cdot \Delta f(x_o) + \frac{1}{2} \cdot s(s-1) \cdot \Delta^2 f(x_o) + $$
$$+ \frac{1}{6} \cdot s(s-1)(s-2) \cdot \Delta^3 f(x_o) + \frac{1}{24} \cdot s(s-1)(s-2)(s-3) \cdot \Delta^4 f(x_o) \qquad (5)$$

The equation can be rearranged as:

$$\begin{array}{lllllllll}
f(x_s) = & 1 & \cdot & (f(x_o) & & & & &) \\
 & + \Delta & \cdot & (& \Delta f(x_o) & - \frac{1}{2}\Delta^2 f(x_o) & + \frac{1}{3}\Delta^3 f(x_o) & - \frac{1}{4}\Delta^4 f(x_o) &) \\
 & + \Delta^2 & \cdot & (& & + \frac{1}{2}\Delta^2 f(x_o) & - \frac{1}{2}\Delta^3 f(x_o) & + \frac{1}{24}\Delta^4 f(x_o) &) \\
 & + \Delta^3 & \cdot & (& & & + \frac{1}{6}\Delta^3 f(x_o) & - \frac{1}{4}\Delta^4 f(x_o) &) \\
 & + \Delta^4 & \cdot & (& & & & + \frac{1}{24}\Delta^4 f(x_o) &)
\end{array} \qquad (6)$$

Or in matrix notation:

$$f(x_s) = (1,\Delta,\Delta^2,\Delta^3\Delta^4 \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{1}{2} & +\frac{1}{3} & -\frac{1}{4} \\ 0 & k & +\frac{1}{2} & -\frac{1}{2} & +\frac{1}{24} \\ 0 & 0 & 0 & +\frac{1}{6} & -\frac{1}{4} \\ 0 & 0 & 0 & 0 & +\frac{1}{24} \end{pmatrix} \cdot \begin{pmatrix} f(x_o) \\ \Delta f(x_o) \\ \Delta^2 f(x_o) \\ \Delta^3 f(x_o) \\ \Delta^4 f(x_o) \end{pmatrix} \qquad (7)$$

or

$$f(x_s) = \bar{S} \cdot A_1 \cdot \bar{D} \qquad (7a)$$

For $m = 4$ Equation (3) can be written:

$$\begin{aligned} f(x_o) &= f(x_o) \\ \Delta f(x_o) &= f(x_1) - f(x_o) \\ \Delta^2 f(x_o) &= f(x_2) - 2f(x_1) + f(x_o) \\ \Delta^3 f(x_o) &= f(x_3) - 3f(x_2) + 3f(x_1) - f(x_o) \\ \Delta^4 f(x_o) &= f(x_4) - 4f(x_3) + 6f(x_2) - 4f(x_1) + f(x_o) \\ &\qquad \ldots\ldots\ldots \qquad (8) \end{aligned}$$

In matrix notation:

$$\begin{pmatrix} f(x_o) \\ \Delta f(x_o) \\ \Delta^2 f(x_o) \\ \Delta^3 f(x_o) \\ \Delta^4 f(x_o) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ -1 & 3 & -3 & 1 & 0 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix} \cdot \begin{pmatrix} f(x_o) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{pmatrix}$$

$$\ldots\ldots\ldots\ldots \qquad (9)$$

or

$$\bar{D} = A_2 \cdot \bar{Y} \qquad (9a)$$

Equation (4) can be written:

$$f(x_s) = \bar{S} \cdot A_1 \cdot A_2 \cdot \bar{Y} \qquad (10)$$

where:

$$A_1 \cdot A_2 = A = \frac{1}{24} \begin{pmatrix} 24 & 0 & 0 & 0 & 0 \\ -50 & 96 & -72 & 32 & -6 \\ 25 & -64 & 54 & -16 & 1 \\ -10 & 36 & -48 & 28 & -6 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix}$$

Thus:

$$f(x_s) = (1, s, s^2, s^3, s^4 \cdot \frac{1}{24} \begin{pmatrix} 24 & 0 & 0 & 0 \\ -50 & 96 & -72 & 32 \\ 25 & -64 & 54 & -16 \\ -10 & 36 & -48 & 28 \\ 1 & -4 & 6 & -4 \end{pmatrix} \begin{pmatrix} 0 & f(x_0) \\ -6 & f(x_1) \\ 1 & f(x_2) \\ -6 & f(x_3) \\ 1 & f(x_4) \end{pmatrix} \qquad (10a)$$

Example: Y = 1, 4, 9, 16, 25

$$f(x_s) = (1, s, s^2, s^3, s^4) \cdot \frac{1}{24} \begin{pmatrix} 24 & 0 & 0 & 0 & 0 \\ -50 & 96 & -72 & 32 & -6 \\ 25 & -64 & 54 & -16 & 1 \\ -10 & 36 & -48 & 28 & -6 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \\ 9 \\ 16 \\ 25 \end{pmatrix} =$$

$$= (1, s, s^2, s^3, s^4) \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 0 \end{pmatrix} = 1 + 2s + s^2 = (1 + s)^2$$

6.2.1 Computer Implementation of a Procedure for Extrapolation and Identification of Noisy Polynomial Sequences

The extrapolator derived in Section 6.2 and there defined by Equation (10) has been implemented in a program for a digital computer. The program is written in IPL-V but could easily be adapted to some algebraic programming language such as FORTRAN. The IPL-V program follows the input- and output-specifications of SEP.

In the programmed version $(x-1)$ has been substituted for x in Equation 5.

$$f(x) = y_1 + (x-1)\Delta y_1 + \frac{1}{2} \cdot (x-1)(x-2) \cdot \Delta^2 y_1 + \frac{1}{6}(x-1)(x-2)(x-3) \cdot \Delta^3 y_1 + \frac{1}{24}(x-1)(x-2)(x-3)(x-4) \quad \Delta^4 y_1 \qquad (5x)$$

Corresponding modifications have been made in Equations 6, 7, and 10.

Certain additional features have been added. So, for instance, only the minimum necessary amount of information about input-sequences is utilized. Therefore, a m:th degree polynomial sequence can be extrapolated from any m+1 consecutive entries, i.e., no minimum number of entries is required. All extrapolations are normalized such that the general expression always uses the first entry of the sequence as origin.

E.g., The sequence: 1, 4, 6, 16, 25, 36, 49, 64, ... if extrapolated from 16, 25, 36 gives the general expression $(x^2 + 6x + 9)$, however a translation of the origin of the general expression is desirable, such that $x = 1$ corresponds to the first value of the sequence. Therefore, in the next step, a sequence for the arguments -2, -1, 0, +1, +2, of x is generated giving the values 1, 4, 9, 16, 25, 36, 49, 64, ... which sequence in turn, is extrapolated from 1, 4, 9, giving the general expression (x^2) . Thus, the normalization is performed without actually employing any particular rules for symbolic transformation of algebraic expressions. A flowchart is given in Figure 6.4.

An actual output from a computer-run of the extrapolator is illustrated in Figure 6.5 A few comments to the example are given below:

Given a sequence of numbers (line 5), SEP on the basis of past experience selects the number of entries from the input-sequence that will be used for forming a temporary hypothesis (line 11). The first four entries are chosen (line 52) and the algorithm is applied (line 71). The general expression $-X^2 + 10X$ is generated (line 73). By substituting the values 1, 2, 3, and 4 for X, four values are generated. These four values are tested against the four input-values (lines 74-77), as the computed values are the same as the given, the complete input-sequence is used to test the hypothesis (lines 78-84). During this test one discrepancy occurs (by previous decisions of SEP one error is permitted), this is ignored and the temporary hypothesis is accepted. Thereafter, an extrapolation of the sequence is made using the temporary hypotheses as generator for values (lines 91-92). The result is stored in the memory for future reference. (line 93)

Examples of extrapolation of "scrambled sequencies" are given in Figure 6.6.

Extrapolation and Identification of Polynomial Sequences

Input of sequence
$Y \underline{def} (y_1, \ldots, y_n)$.

The index j defines the starting point of analysis.

$\bar{Y}$ is defined as a vector of five consecutive entries of the sequence Y .

Equation(10)
$Y_s = d_1 + d_2 \quad x + d_3 \cdot x^2 + d_4 \quad x^3 + d_5 \quad x^4$

NE = the number of discrepancies between Y and its estimate.

Produce estimate of y_i using Y_s .

Test estimate.

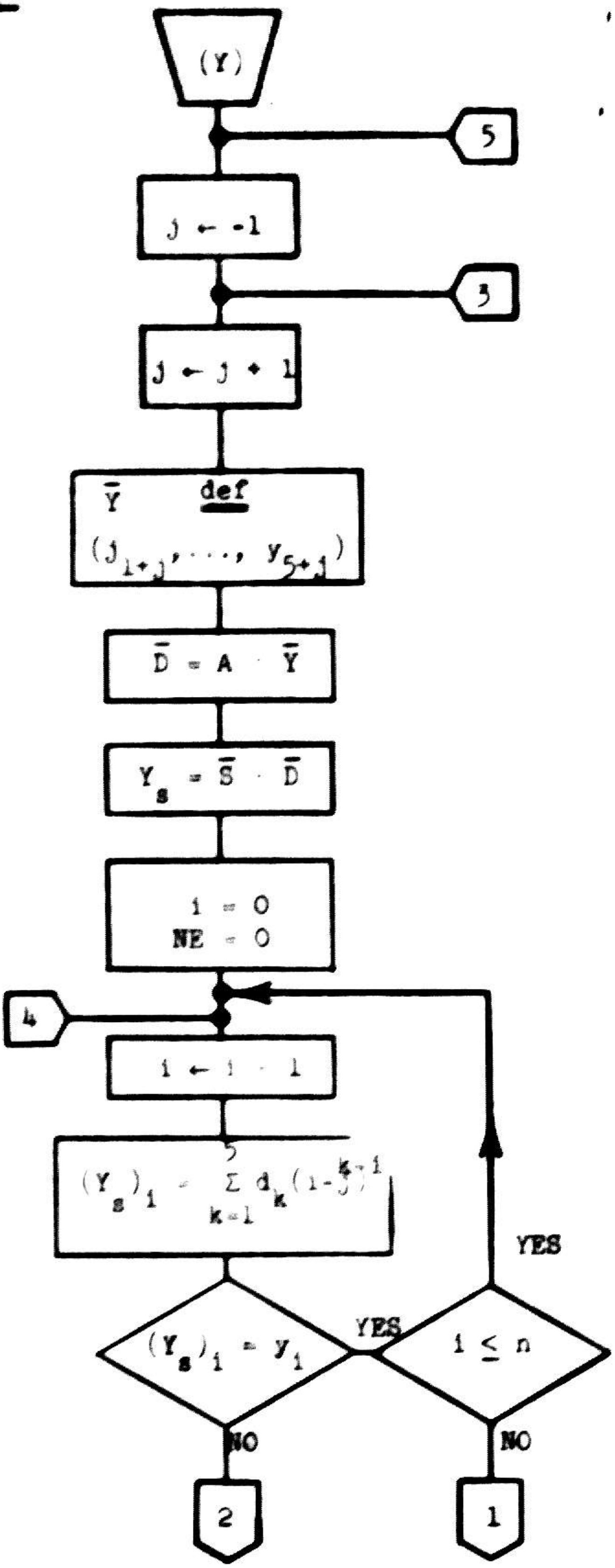
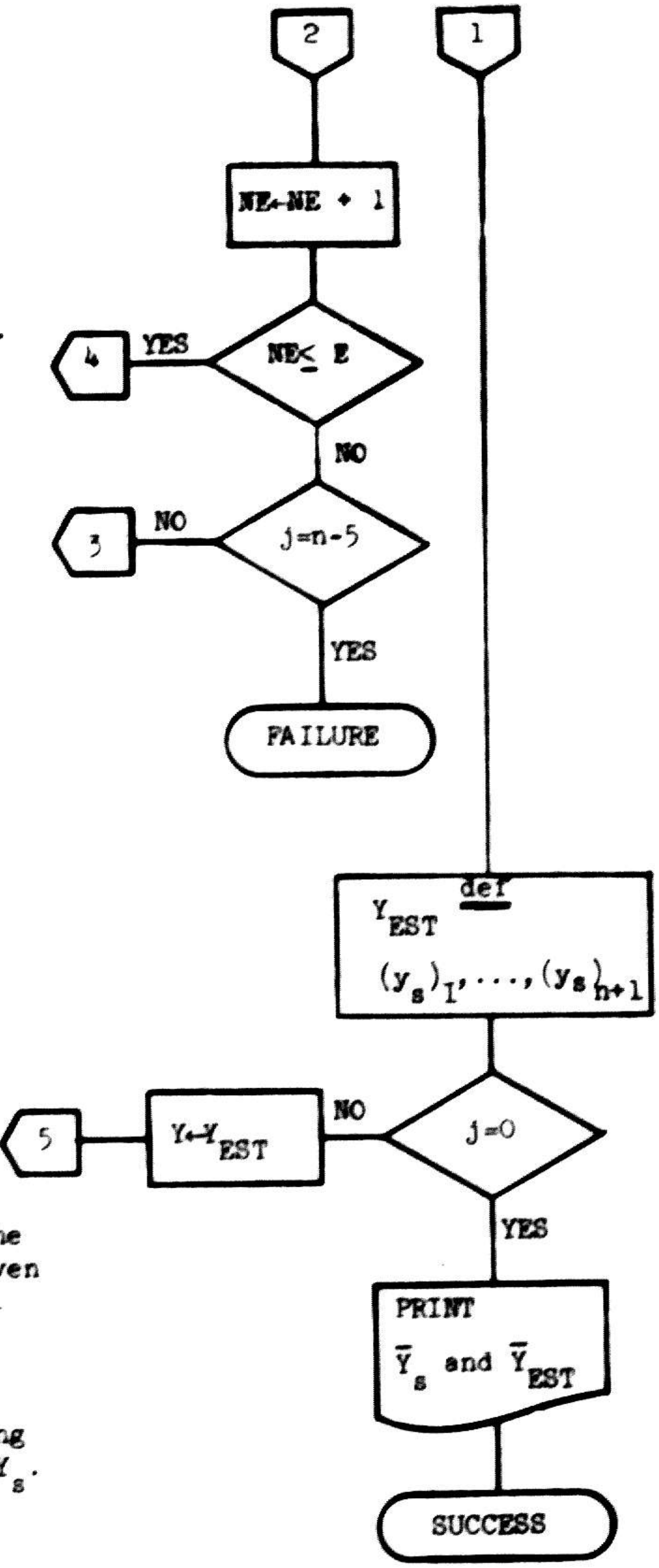

(Y)
5
$j \leftarrow -1$
3
$j \leftarrow j + 1$
$\bar{Y} \underline{def} (j_{1+j}, \ldots, y_{5+j})$
$\bar{D} = A \cdot \bar{Y}$
$Y_s = \bar{S} \cdot \bar{D}$
$i = 0$
$NE = 0$
4
$i \leftarrow i + 1$
$(Y_s)_i = \sum_{k=1}^{5} d_k (i-j)^{k-1}$
YES
$(Y_s)_i = y_i$
YES
$i \leq n$
NO
NO
2
1

Figure 6.4.1


2
1
NE←NE + 1
YES
NE≤ E
4
NO
NO
3
j=n-5
YES
FAILURE
$Y_{EST} \overset{def}{=} (y_s)_1, \ldots, (y_s)_{n+1}$
NO
Y←Y_EST
5
j=0
YES
PRINT
$\overline{Y}_s$ and $\overline{Y}_{EST}$
SUCCESS

4 **Test if number of discrepancies is permissible.**

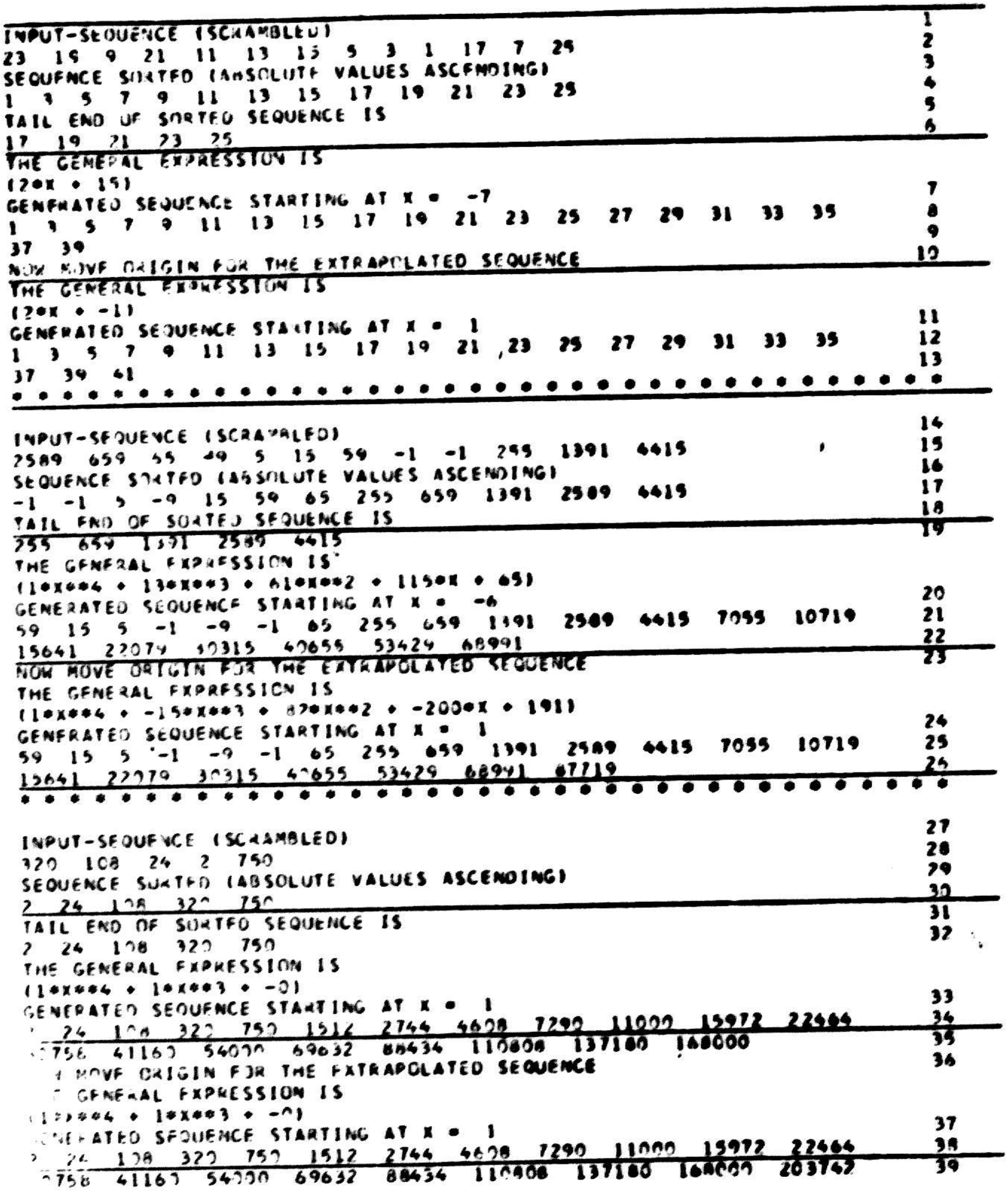
3. All combinations of five consecutive entries of Y has not been tested yet.

5 As there is no provision for translating the origin of a sequence-generating polynomial, the estimated sequence is given as input for the program. This sequence can be extrapolated for $j = 0$, therefore, a correct expression of the generating polynomial is given by Y_s.

Figure 6.4.2

```
NOW TRY PROBLEM NUMBER 7 ...*********************************************
PROCESS 3 CAN NOT BE USED                                                   2
PROCESS 4 CAN NOT BE USED                                                   3

THE INPUT UNDER CONSIDERATION IS                                            4
9  16  21  24  BLANK  24  21  16  9                                         5
ALLOW 1 ERRORS FOR THE INPUT                                                6
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

THE MOST RECENT PROBLEMTYPES WERE                                           7
4  3  1  1  1  2  4  3  1  1  2  2  4  3  1  1  1  2  2  2  4  3            8
                                                                            9
I ASSUME THE NEXT ENTRY WILL BE  1                                         10
NOW TRY PROBLEM-TYPE 1 WITH 4 ENTRIES                                      11
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CURRENT INPUT LIST                                                         52
9  16  21  24


* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THE INPUTLIST IS ONLY  4  ENTRIES LONG SO I TRY POLYNOMIAL ROUTINE        70

USE THE ALGORITHM                                                          71

THE GENERAL EXPRESSION IS                                                  72
(-1*X**2 + 10*X + 0)                                                       73

                  ESTIMATE =  9            INPUT =  9                      74
                  ESTIMATE =  16           INPUT =  16                     75
                  ESTIMATE =  21           INPUT =  21                     76
                  ESTIMATE =  24           INPUT =  24                     77
                  ESTIMATE =  25           INPUT =  BLANK                  78
               SOMETHING IS WRONG HERE                                     79
                  I NOW HAVE 1 DISSIMILARITY(IES)                          80
                  ESTIMATE =  24           INPUT =  24                     81
                  ESTIMATE =  21           INPUT =  21                     82
                  ESTIMATE =  16           INPUT =  16                     83
                  ESTIMATE =  9            INPUT =  9                      84
POLYNOMIAL SEEMS OK                                                        85
THE INPUT TO THE ROUTINE WAS                                               86
9  16  21  24
THE INPUT TO THE PROGRAM WAS                                               88
9  16  21  24  BLANK  24  21  16  9
THE OUTPUT IS                                                              90
9  16  21  24  25  24  21  16  9  0  -11  -24  -39  -56  -75  -96          91
-119  -144  -171  -200
I WILL REMEMBER THIS                                                       93
****************************************************************************
```

Figure 6.5



```
INPUT-SEQUENCE (SCRAMBLED)
23  19  9  21  11  13  15  5  3  1  17  7  25
SEQUENCE SORTED (ABSOLUTE VALUES ASCENDING)
1  3  5  7  9  11  13  15  17  19  21  23  25
TAIL END OF SORTED SEQUENCE IS
17  19  21  23  25
THE GENERAL EXPRESSION IS
(2*X + 15)
GENERATED SEQUENCE STARTING AT X =  -7
1  3  5  7  9  11  13  15  17  19  21  23  25  27  29  31  33  35
37  39
NOW MOVE ORIGIN FOR THE EXTRAPOLATED SEQUENCE
THE GENERAL EXPRESSION IS
(2*X + -1)
GENERATED SEQUENCE STARTING AT X =  1
1  3  5  7  9  11  13  15  17  19  21 ,23  25  27  29  31  33  35
37  39  41
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

INPUT-SEQUENCE (SCRAMBLED)
2589  659  65  -9  5  15  59  -1  -1  255  1391  4415
SEQUENCE SORTED (ABSOLUTE VALUES ASCENDING)
-1  -1  5  -9  15  59  65  255  659  1391  2589  4415
TAIL END OF SORTED SEQUENCE IS
255  659  1391  2589  4415
THE GENERAL EXPRESSION IS
(1*X**4 + 13*X**3 + 61*X**2 + 115*X + 65)
GENERATED SEQUENCE STARTING AT X =  -6
59  15  5  -1  -9  -1  65  255  659  1391  2589  4415  7055  10719
15641  22079  30315  40655  53429  68991
NOW MOVE ORIGIN FOR THE EXTRAPOLATED SEQUENCE
THE GENERAL EXPRESSION IS
(1*X**4 + -15*X**3 + 82*X**2 + -200*X + 191)
GENERATED SEQUENCE STARTING AT X =  1
59  15  5  -1  -9  -1  65  255  659  1391  2589  4415  7055  10719
15641  22079  30315  40655  53429  68991  87719
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

INPUT-SEQUENCE (SCRAMBLED)
320  108  24  2  750
SEQUENCE SORTED (ABSOLUTE VALUES ASCENDING)
2  24  108  320  750
TAIL END OF SORTED SEQUENCE IS
2  24  108  320  750
THE GENERAL EXPRESSION IS
(1*X**4 + 1*X**3 + -0)
GENERATED SEQUENCE STARTING AT X =  1
2  24  108  320  750  1512  2744  4608  7290  11000  15972  22464
30756  41160  54000  69632  88434  110808  137180  168000
NOW MOVE ORIGIN FOR THE EXTRAPOLATED SEQUENCE
THE GENERAL EXPRESSION IS
(1*X**4 + 1*X**3 + -0)
GENERATED SEQUENCE STARTING AT X =  1
2  24  108  320  750  1512  2744  4608  7290  11000  15972  22464
30758  41160  54000  69632  88434  110808  137180  168000  203742
```

Figure 6.6

6.3 Generalized Polynomial Extrapolator

The domain of the basic polynomial extrapolator can be extended by introducing additional operators.

A ℓ:th level sequence $Y^{\ell} = y_0^{\ell}, y_1^{\ell}, \ldots, y_{n-}^{\ell}$ is derived as the result of sequential application of ℓ arbitrary operators O_j $(j = 1, 2, \ldots \ell)$ on a sequence Y.

$$Y^{\ell} = O_{\ell}\,(O_{\ell-1}(O_{\ell-2} \cdots O_1(Y) \ldots)\,)$$

or

$$Y^{\ell} = \pi_{j=1}^{\ell} O_j Y$$

$$Y^{0} = Y \ .$$

The following operators are defined for integer values only.

The difference operator Δ

$$\Delta\, y_i^{\ell} = y_{i+1}^{\ell} - y_i^{\ell} \ .$$

The quotient operator δ

$$\delta y_i^{\ell} = \frac{y_{i+1}^{\ell}}{y_i^{\ell}} \ .$$

The cycle operator $\mathcal{C}^k$

$$\mathcal{C}^k Y^{\ell} \rightarrow y_i^{\ell} = y_{i+j\cdot k}^{\ell} \ .$$

$$i = 0, 1, \ldots, n-\ell$$

$$j = \pm 1, \pm 2, \ldots$$

$$k = 1, 2, \ldots$$

$$\ell = 0, 1, \ldots, n$$

$$1 \leq i + jk \leq n-\ell \ .$$

Inverse application of the operators can be used for extrapolation

$$\Delta^{-}(\Delta y_n^{\ell}, y_n^{\ell}) = y_{n+1}^{\ell} = \Delta y_n^{\ell} + y_n^{\ell}$$

$$\delta^{-}(\delta y_n^{\ell}, y_n^{\ell}) = y_{n+1} = \delta y_n^{\ell} \cdot y_n^{\ell}$$

$$(\not{c}^{k})^{-} Y^{\ell} \rightarrow y_{n+1}^{\ell} = y_{n-k+1}^{\ell} .$$

At each level of analysis the operators are applied in order $\not{c}$, δ , and Δ because:

> $\not{c}$ is used as stopping rule for recursion
>
> Δ applies to any sequence, therefore δ would never be applied if preceded by Δ .

Extrapolation proceeds as follows:

1. Operators are applied recursively on sequences Y^{ℓ} until any one of two stopping-rules is met.

   a. The sequence Y^{n} is generated,[5] or

   b. The $\not{c}$ operator applies.

   Thus, at most, n applications of operators leads to termination.

2. Application of all employed operators in reverse order provides desired extrapolation.

The domain of the present extrapolator is rather wide.[6] By the difference-operator, all polynomial sequences can be extrapolated. The quotient operator is applicable on exponential and factorial sequences. The cycle-operator by itself can recognize all cyclical sequences and, in combination with the other operators, recognizes intertwined sequences. Other combinations of operators cover rather complicated generating expressions; however, as illustrated in Section 4.1, the present extrapolator is not selective enough to be employed by SEP.

6.3.1 Computer-Implementation of a Generalized Polynomial Extrapolator

The generalized polynomial extrapolator is, at present, not included in the complex sequence-extrapolator SEP, the reason being its generality which would "overlook" available contextual information and thus decrease the power of the compound. An implementation, currently programmed in FORTRAN, is illustrated as a flow-chart in Figure 6.7, here recursion is simulated by iteration.

6.3.2 Examples of Representation

Examples 4, 5, and 6 of Section 4.1 are below represented as is implicitly done by the Generalized Polynomial Extrapolator. The ith entry of a sequence is denoted n_i.

Example 4: 1 10 100 1000 10000

δ: 10 10 10 10

$$\delta^{-}(\delta n_{i-1}, n_{i-1}) = n_i = \delta n_{i-1} \cdot n_{i-1} = 10 \cdot n_{i-1}$$

The representation is

$$n_i = 10 \cdot n_{i-1}$$

Example 5: 1 11 111 1111 11111

Δ: 10 100 1000 10000

δ: 10 10 10

Generalized Polynomial Extrapolator

Input sequence

$Y^{\ell} = y_1, y_2, \ldots, y_n$.

$\mathcal{L}^k$ applies when $y_i^{\ell} = y_{i+k}^{\ell}$ for all i, $i = 1,2,\ldots, n-k-\ell$

δ applies when for all i, $i = 1,2,\ldots, n-1-\ell$ y_i is a factor c y_{i+1} .

Δ always applies.

Apply operator 0_{ℓ} (Push down previous operator).

The last entry of Y^{ℓ} is $y^{\ell}_{n-\ell}$, as $y_i^{\ell} = y_{i+k}^{\ell}$ the next entry is $y^{\ell}_{n-\ell+1} = y^{\ell}_{n-\ell+1-k}$.

Apply all inverse operators in reverse order of application. (Pop up the operators).

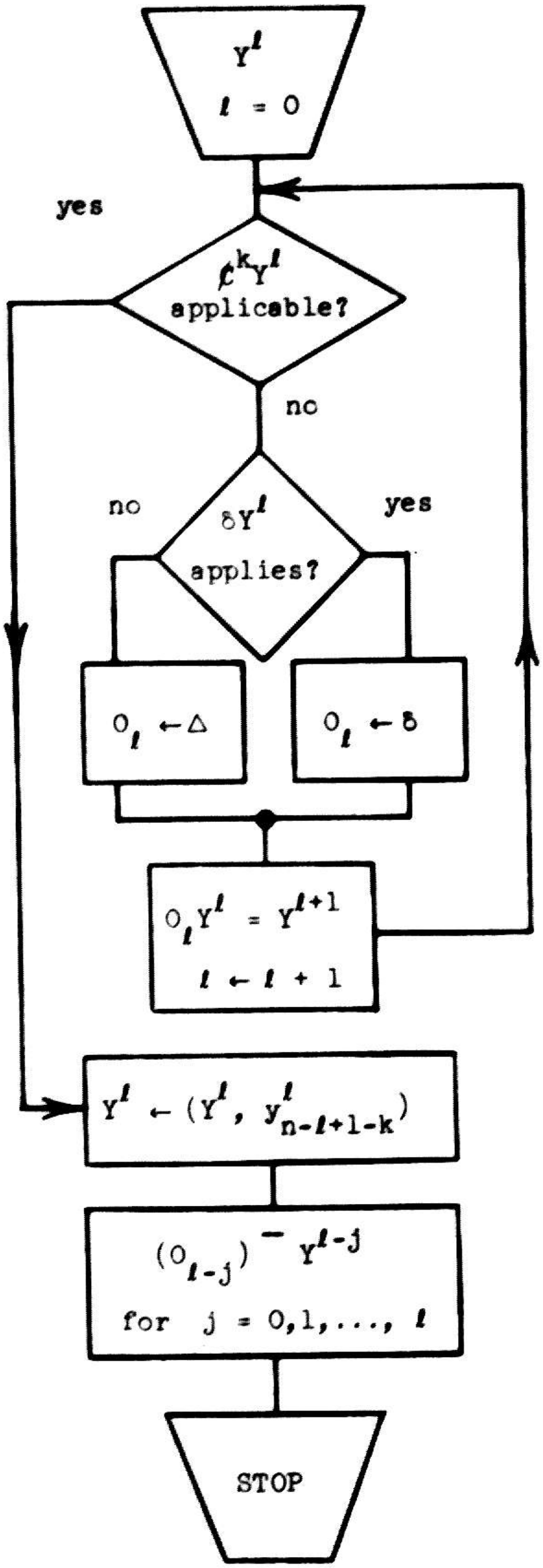

Y^{ℓ}
$\ell = 0$
yes
$\mathcal{L}^k Y^{\ell}$ applicable?
no
no
δY^{ℓ} applies?
yes
$0_{\ell} \leftarrow \Delta$
$0_{\ell} \leftarrow \delta$
$0_{\ell} Y^{\ell} = Y^{\ell+1}$
$\ell \leftarrow \ell + 1$
$Y^{\ell} \leftarrow (Y^{\ell}, y^{\ell}_{n-\ell+1-k})$
$(0_{\ell-j})^{-} Y^{\ell-j}$
for $j = 0,1,\ldots,\ell$
STOP

Figure 6.7

From the representation of example 4, we have

$$\Delta n_i = \delta(\Delta n_{i-1}) \cdot (\Delta n_{i-1}) = 10 \cdot \Delta n_{i-1}$$

$$\Delta n_i = n_i - n_{i-1}$$

$$(n_i - n_{i-1}) = 10(n_{i-1} - n_{i-2})$$

$$\begin{aligned} n_i &= 10\, n_{i-1} + (n_{i-1} - 10\, n_{i-2}) \\ &= 10\, n_{i-1} + (10\, n_{i-2} + n_{i-2} - 10\, n_{i-3} - 10\, n_{i-2}) \\ &= 10\, n_{i-1} + (n_{i-2} - 10\, n_{i-3}) \end{aligned}$$

In general we can derive

$$n_i = 10\, n_{i-1} + (n_k - 10\, n_{k-1}) \quad k = 1,2, \ldots$$

or

$$n_i - 10\, n_{i-1} = n_k - 10\, n_{k-1}$$

which holds for all k

For k=2 we have

$$n_i - 10\, n_{i-1} = 11 - 10 \cdot 1 = 1$$

The representation is

$$n_i = 10\, n_{i-1} + 1$$

Example 6: 1 12 123 1234 12345 123456

Δ 11 111 1111 11111 111111

Using the representation of example 5

$$\Delta n_i = 10\, \Delta n_{i-1} + 1$$

$$n_i - n_{i-1} = 10\,(n_{i-1} - n_{i-2}) + 1$$

$$n_i = 10\, n_{i-1} + (n_{i-1} - 10\, n_{i-2} + 1)$$

$$n_i = 10\, n_{i-1} + (10\, n_{i-2} + n_{i-2} - 10 n_{i-3} + 1 - 10 n_{i-2} + 1)$$

$$n_i = 10\, n_{i-1} + (n_{i-2} - 10\, n_{i-3} + 2)$$

In general

$$n_i = 10\ n_{i-1} + (n_k - 10\ n_{k-1} + i-k)$$

or

$$n_i - 10\ n_{i-1} - i = n_k - 10\ n_{k-1} - k$$

Using the initial conditions for k = 3

we have

$$n_i - 10\ n_{i-1} - i = 123 - 10 \cdot 12 - 3 = 0$$

The representation is

$$n_i = 10\ n_{i-1} + i$$

6.4 An Extended Sequence-extrapolator

The domain A^g of the present extrapolator is recursively defined.[7]

$$A^g = A^{g-1} \cdot B^{g^{C^g}}$$

$$A^o \in P^k$$

$$B^i \in P^\ell$$

$$C^j \in P^m$$

P^h = The set of all polynomials of degree h or less which have integer coefficients.

$$g = 1,2,\ldots$$

$$h = 0,1,\ldots$$

$$i = 1,2,\ldots,n$$

$$j = 1,2,\ldots,n$$

$$k = 0,1,\ldots$$

$$\ell = 0,1,\ldots$$

$$m = 0,1,\ldots$$

A sequence Y belonging to the domain A^g can, in general, not be extrapolated by strictly algorithmic methods. Repeated application of polynomial extrapolators to sequences derived from Y will, however, suffice for its extrapolation.

The application of polynomial extrapolators requires that each entry y_i of the sequence Y is represented as three separate elements a_i, b_i, and c_i .

$$y_i = a_i \cdot b_i^{c_i} .$$

If this is the case, the three sequences

$$A = a_o, a_1, \ldots, a_n$$

$$B = b_o, b_1, \ldots, b_n$$

$$C = c_o, c_1, \ldots, c_n$$

can be separately extrapolated.

The subdivision of Y into A, B, and C is, of course, no trivial task. The limitation of the domains of the component sequences, however, permits rather efficient search for proper subdivisions.

By definition

$$y_i = a_i \cdot b_i^{c_i} = \underbrace{a_i \cdot b_i \cdot \ldots \cdot b_i}_{c_i \text{ repetitions of } b_i}$$

The search for subsequencies A, B, C now proceeds in three phases.

Phase 1:

Find sequence C.

From a given number y_i, it is not possible to directly identify corresponding c_i . An upper bound can, however, be found because a prime factor occurring X times in a_i and Z times in b_i must be represented

$$X + c_i \cdot Z \text{ times in } y_i .$$

Result 1. The exponent c_i can not assume any value greater than the number of occurrencies of the most frequent prime factor of y_i .[8]

Step 1. Generate a sequence $P = p_o,., p_i,.., p_n$ such that for each $y_i \in Y$, p_i is the number of occurrencies of its most frequently occurring prime factor.[9]

By Result 1, the relation $c_i \leq p_i$ in general holds.[10] Therefore, P may be considered as an upper bound for C.

Step 2. Find a sequence $C_{est} \in P^m$ which is bounded from above by the sequence P .

We have assumed that $m = 1$, thus C is linear and its estimate may be written

$$C_{est} = k \cdot i + h \qquad .$$

C_{est} can not be derived directly from P, however, a rather efficient search-procedure can be designed for its identification. The search for C_{est} is most easily conceptualized as the fitting of a line to a set of points (i, p_i) in a 2-dimensional graph.

Result 1 implies the following restriction:

Restriction 1: C_{est} may, for at most $m = 2 + E$ arguments i, exceed corresponding P. (E stands for the maximum number of irregularities in Y .)[11]

Any estimate which satisfies Restriction 1 is said to be legal.

The following heuristic rules for choice of parameters for C_{est} are employed.

1. The initial estimates of h and k, denoted by h^o and k^o, are set as follows:

$$h^o = P_o$$

$$k^o = \frac{P_{i*} - P_o}{i*}$$

$i*$ is the largest i for which the inequality $P_i \leq P_{i-1}$ holds.

2. If $C_{est} = k \cdot i + h$ satisfies Restriction 1, then k and/or h are incremented by 1 until neither can be incremented without violating Restriction 1. Then Rule 3 is applied.

   If C_{est} does not satisfy Restriction 1, k and/or h are decreased in the following order until the resulting sequence is legal.

   i. k is decreased in steps of 1.

   ii. If $k = 0$, h is decreased by 1 and the initial value of k is restored.

   iii. The procedure is continued until C_{est} is legal, then Rule 2 is applied.

3. The "highest legal" estimate is: $C_{est} = k \cdot i + h$.

Assumption 1: C_{est} is a correct estimate of C .

Phase 2:

Find sequence B .

In the relation $y_i = a_i \cdot b_i^{c_i}$, a given value of c_i implies that each factor of b_i occurs at least c_i times in y_i . Thus, no prime factor occurring less than c_i times in y_i can be contributed by b_i .

Result 2. b_i cannot exceed the product of all prime factors which occur at least c_i times in y_i .[12]

Step 1. Generate a sequence $Q = q_o, \cdot, q_i, \ldots, q_n, \ldots$ such that for each $y_i \in Y$, q_i is the product of the r:th power of every prime factor which occurs at least $r \cdot c_i$ times in y_i . (r is a positive integer)

By Result 2, the relation $b_i \leq q_i$, in general, holds. Therefore, Q may be considered as an upper bound of B.

We have assumed that $B \in P^1$, therefore, substitution of B for C and Q for P will permit application of Step 2 of Phase 1 to find $B_{est} \in P^1$

Assumption 2: B_{est} is a correct estimate of B.

Phase 3:

Identify the sequence A by the definition

$$y_i = a_i \cdot b_i^{c_i} \quad \text{or} \quad a_i = \frac{y_i}{b_i^{c_i}} .$$

Step 1. Generate a sequence $A = a_o, a_1, \ldots, a_n$ such that for each $y_i \in Y$

$$a_i = \frac{y_i}{b_i^{c_i}}$$

Step 2. A is tested by the polynomial extrapolator. If $A \notin P^4$, the extended extrapolator is applied (substituting A for Y). When, at some level of recursion $A \in P^4$, corresponding sequence Y can be extrapolated by parts A, B, and C. This permits extrapolation of the next level, etc., until the original sequence Y is extrapolated.

6.4.1 Computer-Implementation of an Extended Sequence Extrapolator

At present, a somewhat simplified version of the extended extrapolator is included in SEP. The domain can be written

$$N = A \cdot B^C$$

$$A \in P^4$$

$$B \in P^1$$

$$C \in P^1$$

The program written in IPL-V is completely compatible with other members of SEP. Some ambiguities of the procedure have had to be resolved in programming. A few of these are exemplified:

Examples:

Suppose that the sequence $C = 1,1,1,1,1\ldots\ldots$ i.e., $c_i = 1$ for all i. Then it is impossible to isolate b_i from a_i on the basis of the number of prime factors in n_i. Therefore, when a is a polynomial sequence of 4:th order and b is of first order the resulting sequence $a \cdot b$ is of the 5:th order, a case which is not included in the domain of the polynomial extrapolator.

The easiest way out of this difficulty is to simply extend the domain of the polynomial extrapolator to include 5:th order polynomials. Although this extension is easily implemented, we have chosen another approach, namely, to use a simple heuristic for guessing the form of the sequence b.

Write b_i^1 as $L + (i-1) \cdot 1$ set L to the value of one of the prime factors of the first entry of the input-sequence. If for every i [13] $\frac{n_i}{L+(i-1)}$ is integer, then we assume that a_i = the value of said integer.

The method as outlined seems to cover a very restricted class of situations (particularly as the coefficient for $(i-1)$ is $=1$). We can, however, show that the procedure may be applied in more general situations,[14] because:

Assume $b_i = L + (i-1) \cdot K$ then $n_i = a_i \cdot (L + (i-1) \cdot K)$ or $n_i = K \cdot a_i \left(\frac{L}{K} + (i-1) \right)$, where b_i is of the same form as above, and where the sequence $K \cdot a$ can be extrapolated by the polynomial routine equally well as a itself, provided that $\frac{L}{K}$ is integer-valued.

Several other types of difficulties may occur in the process of extrapolation. So, for instance, values $n_i = 0$ can in general not be accepted, because <u>if</u> the exponential part $b_i^{c_i}$ would turn out to be 0 (i e., $b_i = 0$), we would have to divide n_i by 0 in order to identify the value of a_i The solution to this problem is very simple; delete 0 if it occurs in the first position of the input-sequence, otherwise replace it with 1 (this will, of course, introduce an error into the input-sequence, but by adding one to the number of permissable errors this will be taken care of).

In addition to above mentioned instances, several local tests and safeguards against adverse combinations of parameters have been included into the sequence-extrapolator whenever deemed necessary.

Representation of Results

The procedure used for identification of the sequences B and C will implicitly factor out powers of sub-expressions of 0:th or 1:st degree. The actual result of the factoring will depend upon certain adaptive parameters so, for instance, the polynomial:

$$(X^4 - X^2)$$

may be printed as:

$$(X^2 - 1) \cdot X^2 \;;$$
$$(X^3 - X) \cdot X \;;$$
$$(X^3 + X^2) \cdot (X - 1) \;; \text{ or}$$
$$(X^3 - X^2) \cdot (X + 1) \;.$$

By applying the procedure recursively, we could also get:

$$(X + 1) \cdot (X - 1) \cdot X^2 \;;$$
$$(X^2 + X) \cdot (X - 1) \cdot X \;; \text{ or}$$
$$(X^2 - X) \cdot (X + 1) \cdot X \;; \text{ etc.}$$

The general expression is internally represented as a list of nine parameters (A_4, A_3, A_2, A_1, A_o, B_1, B_o, C_1, and C_o) which represent:

$$(A_4X^4 + A_3X^3 + A_2X^2 + A_1X + A_o) \cdot (B_1X + B_o)^{(C_1X + C_o)} \;.$$

The above general expression will be simplified as much as possible before it is printed out, some rules are:

a. A term A_iX^i where $A_i = 0$ will not be printed out,

e.g., $(5X^4 + 0X^3 + 2X^2 + 0X -2)$ is printed:

$(5X^4 + 2X^2 - 2)$.

b. If $(B_1X + B_o) = 1$, then the exponential part will be deleted,

e.g., $(5X^4 + 2X^2 - 2) \cdot (0X + 1)^{(5X + 2)}$ is printed:

c. If $(C_1X + C_o) = 1$, then the exponent will be deleted,

e.g., $(5X^4 + 2X^2 - 2) \cdot (5X + 2)^{(0X + 1)}$ is printed:

$(5X^4 + 2X^2 - 2) \cdot (5X + 2)$.

d. If $(C_1X + C_o) = 0$, then the exponential part will be deleted,

e.g., $(5X^4 + 2X^2 - 2) \cdot (5X + 2)^{(0X + 0)}$ is printed:

$(5X^4 + 2X^2 - 2)$

Error-Correction by the Extended Sequence-Extrapolator

It can be shown that all error-correction features available for the polynomial extrapolator are also applicable to the extended extrapolator. Intuitively we can consider the present extrapolator as an extension of its polynomial counter-part, where the added features (extrapolation of the exponential part) do not reduce the error-correction abilities. To show how the most complicated case can be taken care of, let us study how a "scrambled" sequence, if necessàry, can be "unscrambled".

From the form of the general expression, we have:

(4:th order polynomial) · (exponential expression) .

The exponential expression can be considered as a transforming operator. There are basically two different situations:

1. The transform is monotonous.
2. The transform gives rise to oscillations.

The two cases are illustrated in Figure 6.8.

The illustration of the monotonous case shows that the transformed polynomial has actually lost the two original extremum-points, which permits us to set X* to a lower value than in the pure polynomial case (in our situation X* = 1). By decreasing the value of X*, we have increased the possibility of finding a "tail" of K+1 entries,[15] thus, also increased the power of the error-correction facilities.

The oscillatory case may seem somewhat more difficult, but by essentially splitting the sequence in two parts, one for even and the other for odd indices, the procedure ends up being identical with the previous case. It is also possible to identify a "tail" for the oscillating sequence by studying only absolute-values, which in the "tail" are increasing monotonously.

A simplified flow-chart over the extrapolator is given in Figure 6.9.

In Figure 6.10, an example of an actual output from a computer-run of the extrapolator is given. The example is self-documenting, therefore, no comments will be given here.

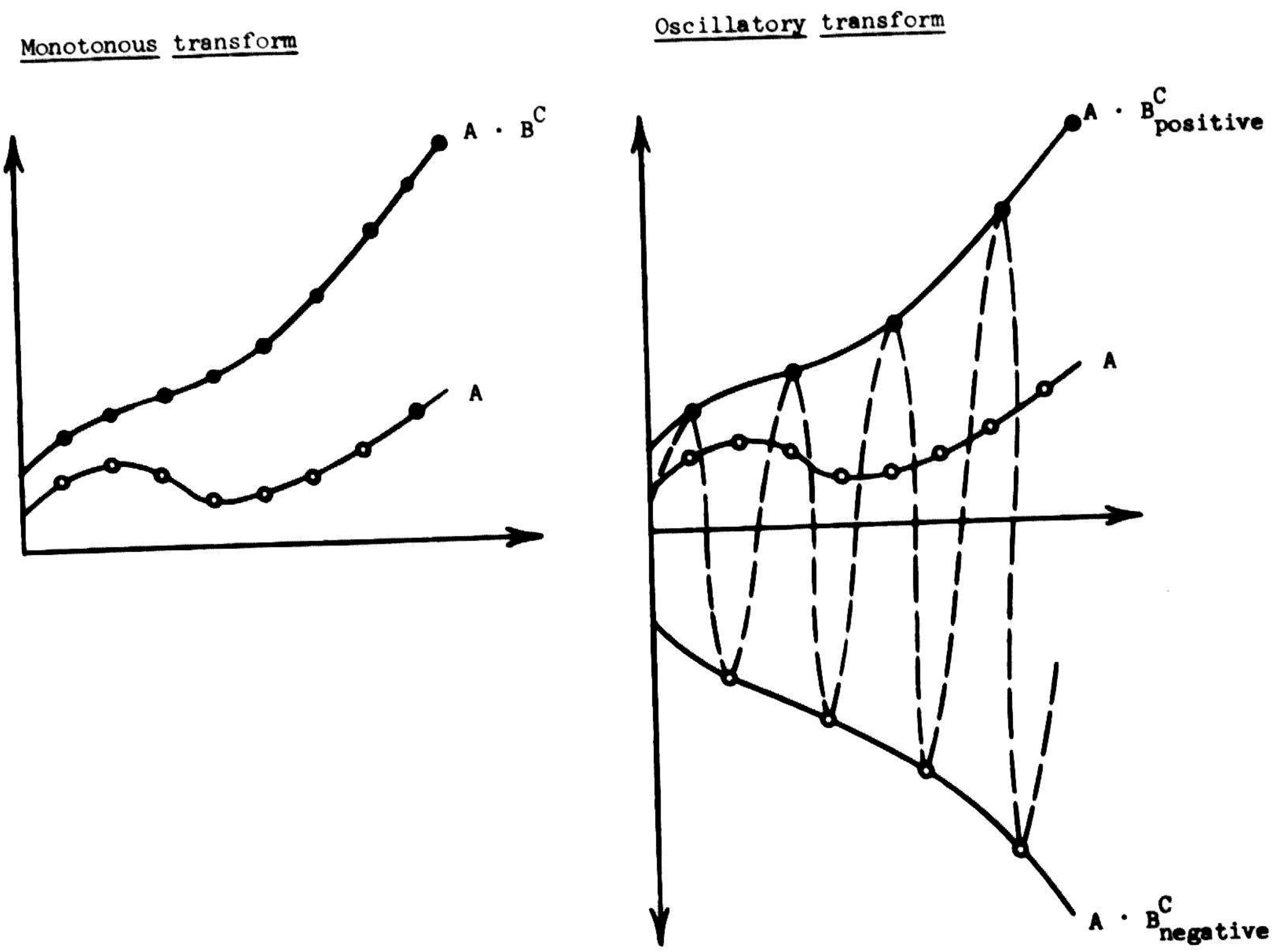

Monotonous transform
$A \cdot B^C$
A
Oscillatory transform
$A \cdot B^C_{positive}$
A
$A \cdot B^C_{negative}$

Figure 6.8

Simplified Flow-Chart of Extended Sequence Extrapolator

Y_1^n = a list of n numbers.

$(Y)_1^n$ = each number of a list.

$Y_1^n \to Y_1^{n+1}$ = extrapolation.

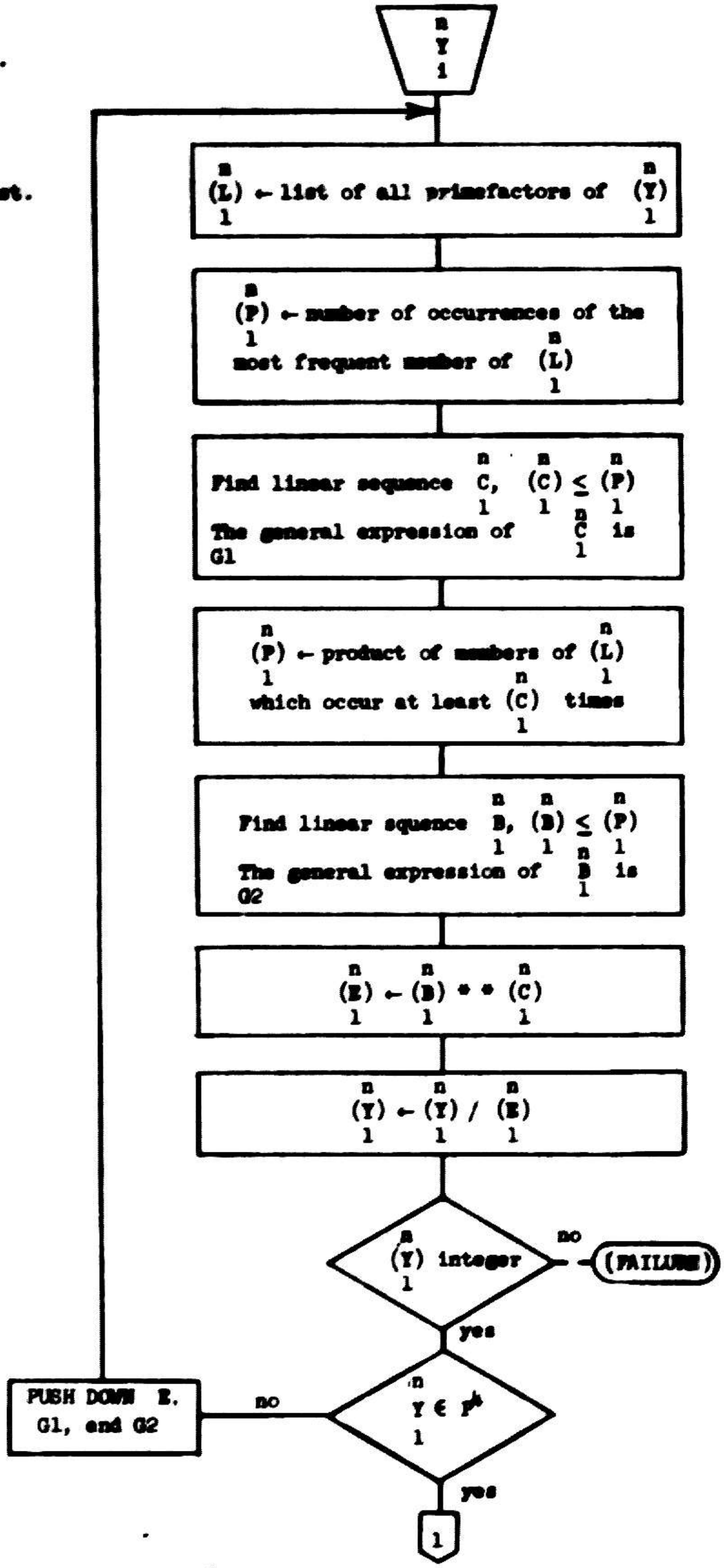

Y_1^n
$(L)_1^n$ ← list of all primefactors of $(Y)_1^n$
$(P)_1^n$ ← number of occurrences of the most frequent number of $(L)_1^n$
Find linear sequence C_1^n, $(C)_1^n \leq (P)_1^n$
The general expression of C_1^n is G1
$(P)_1^n$ ← product of numbers of $(L)_1^n$ which occur at least $(C)_1^n$ times
Find linear squence B_1^n, $(B)_1^n \leq (P)_1^n$
The general expression of B_1^n is G2
$(E)_1^n \leftarrow (B)_1^n ** (C)_1^n$
$(Y)_1^n \leftarrow (Y)_1^n / (E)_1^n$
$(Y)_1^n$ integer
no
(FAILURE)
yes
$Y_1^n \in P^k$
no
PUSH DOWN E, G1, and G2
yes
1

Figure 6.9.1

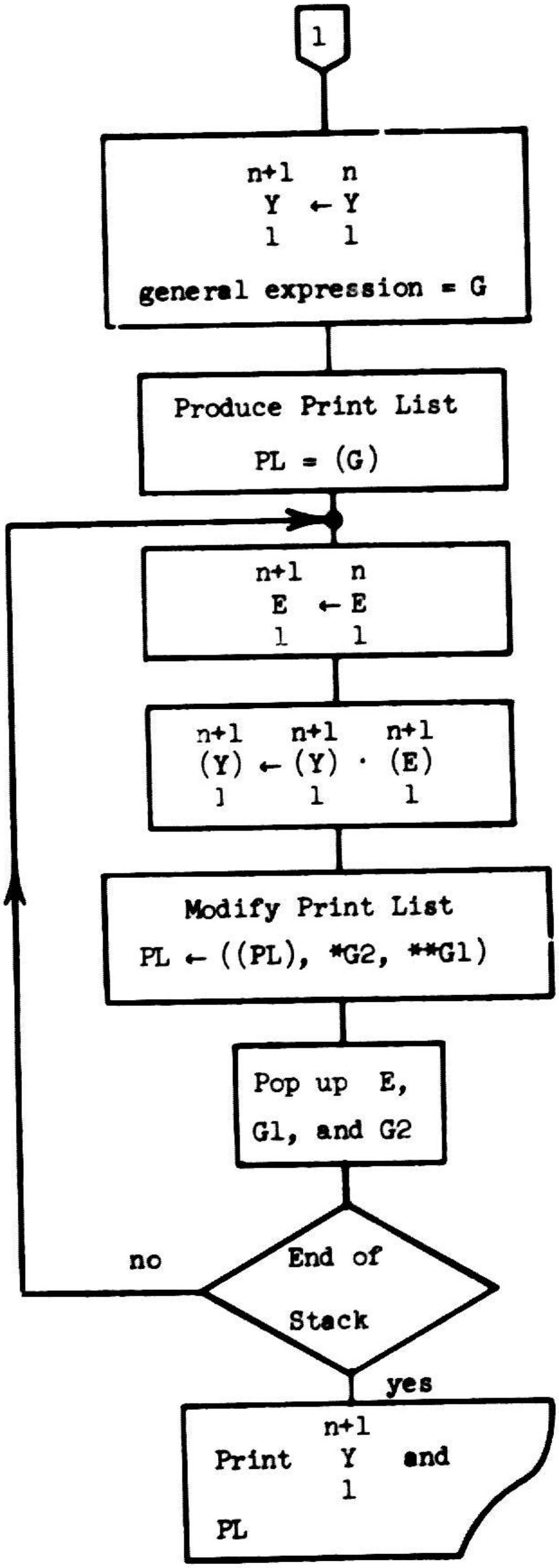
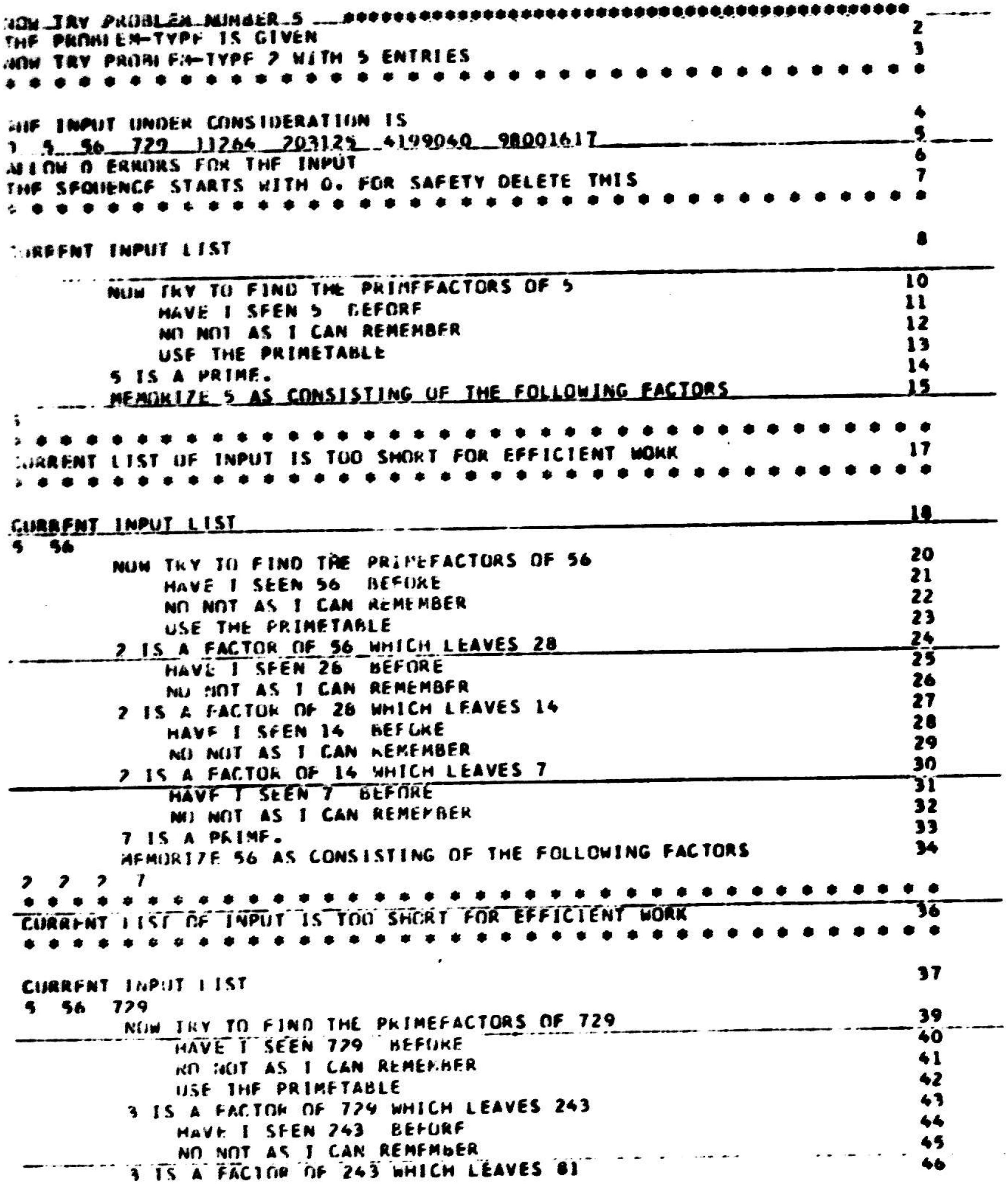
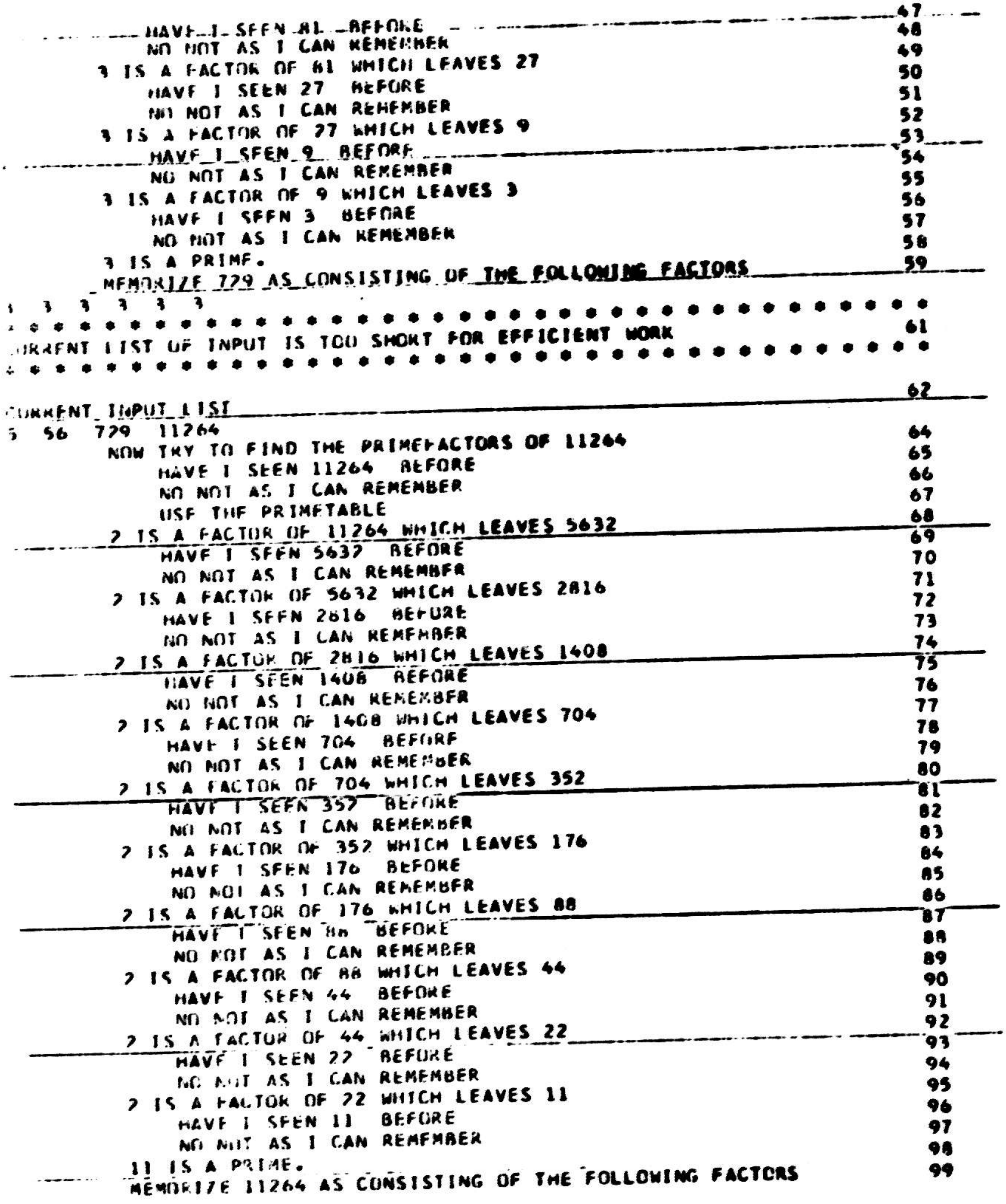

1
n+1 n
Y ← Y
1 1
general expression = G
Produce Print List
PL = (G)
n+1 n
E ← E
1 1
n+1 n+1 n+1
(Y) ← (Y) · (E)
1 1 1
Modify Print List
PL ← ((PL), *G2, **G1)
Pop up E,
G1, and G2
End of
Stack
no
yes
n+1
Print Y and
1
PL

Figure 6.9.2



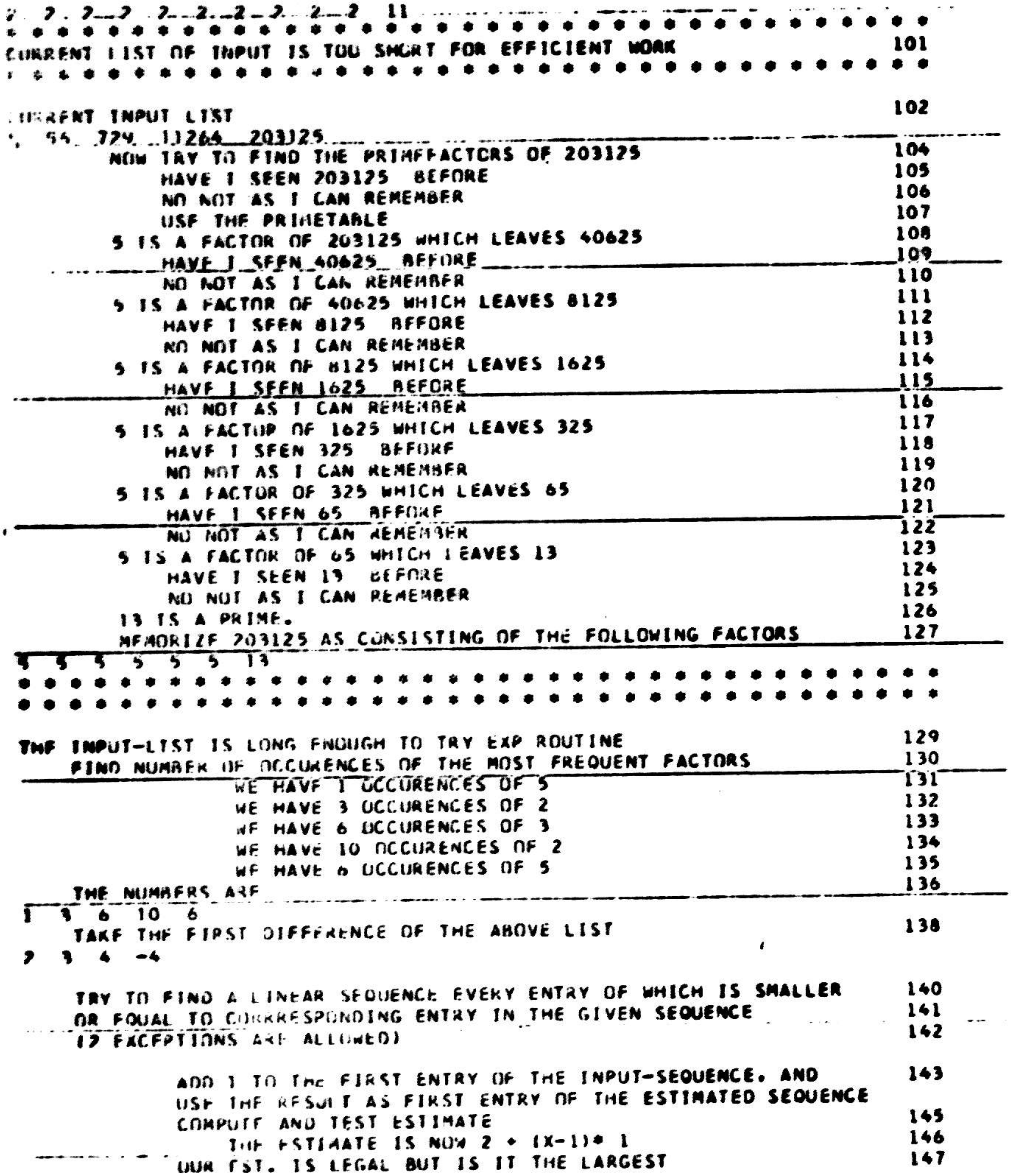
```
NOW TRY PROBLEM NUMBER 5 ****************************************************
THE PROBLEM-TYPE IS GIVEN                                                   2
NOW TRY PROBLEM-TYPE 2 WITH 5 ENTRIES                                       3
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

THE INPUT UNDER CONSIDERATION IS                                            4
0 5 56 729 11264 203125 4199040 98001617                                    5
ALLOW 0 ERRORS FOR THE INPUT                                                6
THE SEQUENCE STARTS WITH 0. FOR SAFETY DELETE THIS                          7
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CURRENT INPUT LIST                                                          8

        NOW TRY TO FIND THE PRIMEFACTORS OF 5                               10
            HAVE I SEEN 5  BEFORE                                           11
            NO NOT AS I CAN REMEMBER                                        12
            USE THE PRIMETABLE                                              13
        5 IS A PRIME.                                                       14
        MEMORIZE 5 AS CONSISTING OF THE FOLLOWING FACTORS                   15
5
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
CURRENT LIST OF INPUT IS TOO SHORT FOR EFFICIENT WORK                       17
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CURRENT INPUT LIST                                                          18
5  56
        NOW TRY TO FIND THE PRIMEFACTORS OF 56                              20
            HAVE I SEEN 56  BEFORE                                          21
            NO NOT AS I CAN REMEMBER                                        22
            USE THE PRIMETABLE                                              23
        2 IS A FACTOR OF 56 WHICH LEAVES 28                                 24
            HAVE I SEEN 26  BEFORE                                          25
            NO NOT AS I CAN REMEMBER                                        26
        2 IS A FACTOR OF 26 WHICH LEAVES 14                                 27
            HAVE I SEEN 14  BEFORE                                          28
            NO NOT AS I CAN REMEMBER                                        29
        2 IS A FACTOR OF 14 WHICH LEAVES 7                                  30
            HAVE I SEEN 7  BEFORE                                           31
            NO NOT AS I CAN REMEMBER                                        32
        7 IS A PRIME.                                                       33
        MEMORIZE 56 AS CONSISTING OF THE FOLLOWING FACTORS                  34
2  2  2  7
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
CURRENT LIST OF INPUT IS TOO SHORT FOR EFFICIENT WORK                       36
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CURRENT INPUT LIST                                                          37
5  56  729
        NOW TRY TO FIND THE PRIMEFACTORS OF 729                             39
            HAVE I SEEN 729  BEFORE                                         40
            NO NOT AS I CAN REMEMBER                                        41
            USE THE PRIMETABLE                                              42
        3 IS A FACTOR OF 729 WHICH LEAVES 243                               43
            HAVE I SEEN 243  BEFORE                                         44
            NO NOT AS I CAN REMEMBER                                        45
        3 IS A FACTOR OF 243 WHICH LEAVES 81                                46
```

Figure 6.10.1



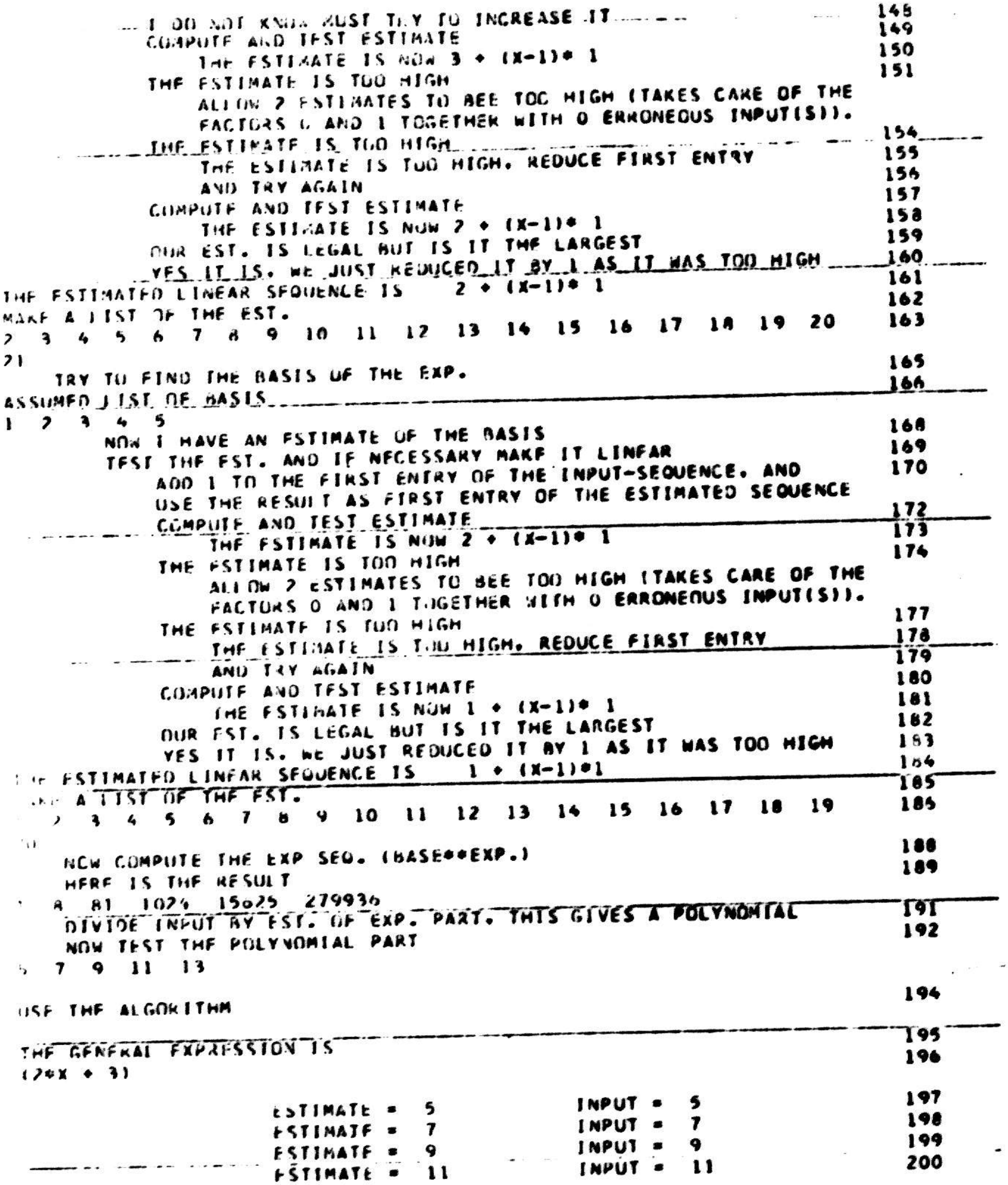
```
            HAVE I SEEN 81  BEFORE
            NO NOT AS I CAN REMEMBER
        3 IS A FACTOR OF 81 WHICH LEAVES 27
            HAVE I SEEN 27  BEFORE
            NO NOT AS I CAN REMEMBER
        3 IS A FACTOR OF 27 WHICH LEAVES 9
            HAVE I SEEN 9  BEFORE
            NO NOT AS I CAN REMEMBER
        3 IS A FACTOR OF 9 WHICH LEAVES 3
            HAVE I SEEN 3  BEFORE
            NO NOT AS I CAN REMEMBER
        3 IS A PRIME.
        MEMORIZE 729 AS CONSISTING OF THE FOLLOWING FACTORS
3  3  3  3  3  3
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
CURRENT LIST OF INPUT IS TOO SHORT FOR EFFICIENT WORK
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CURRENT INPUT LIST
5  56  729  11264
        NOW TRY TO FIND THE PRIMEFACTORS OF 11264
            HAVE I SEEN 11264  BEFORE
            NO NOT AS I CAN REMEMBER
            USE THE PRIMETABLE
        2 IS A FACTOR OF 11264 WHICH LEAVES 5632
            HAVE I SEEN 5632  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 5632 WHICH LEAVES 2816
            HAVE I SEEN 2816  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 2816 WHICH LEAVES 1408
            HAVE I SEEN 1408  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 1408 WHICH LEAVES 704
            HAVE I SEEN 704  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 704 WHICH LEAVES 352
            HAVE I SEEN 352  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 352 WHICH LEAVES 176
            HAVE I SEEN 176  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 176 WHICH LEAVES 88
            HAVE I SEEN 88  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 88 WHICH LEAVES 44
            HAVE I SEEN 44  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 44 WHICH LEAVES 22
            HAVE I SEEN 22  BEFORE
            NO NOT AS I CAN REMEMBER
        2 IS A FACTOR OF 22 WHICH LEAVES 11
            HAVE I SEEN 11  BEFORE
            NO NOT AS I CAN REMEMBER
        11 IS A PRIME.
        MEMORIZE 11264 AS CONSISTING OF THE FOLLOWING FACTORS
```

Figure 6.10.2



```
2  2. 2  2  2  2  2  2  2  2  11
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
CURRENT LIST OF INPUT IS TOO SHORT FOR EFFICIENT WORK              101
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CURRENT INPUT LIST                                                 102
1  55  729  11264  203125
     NOW TRY TO FIND THE PRIMEFACTORS OF 203125                    104
         HAVE I SEEN 203125  BEFORE                                105
         NO NOT AS I CAN REMEMBER                                  106
         USE THE PRIMETABLE                                        107
     5 IS A FACTOR OF 203125 WHICH LEAVES 40625                    108
         HAVE I SEEN 40625  BEFORE                                 109
         NO NOT AS I CAN REMEMBER                                  110
     5 IS A FACTOR OF 40625 WHICH LEAVES 8125                      111
         HAVE I SEEN 8125  BEFORE                                  112
         NO NOT AS I CAN REMEMBER                                  113
     5 IS A FACTOR OF 8125 WHICH LEAVES 1625                       114
         HAVE I SEEN 1625  BEFORE                                  115
         NO NOT AS I CAN REMEMBER                                  116
     5 IS A FACTOR OF 1625 WHICH LEAVES 325                        117
         HAVE I SEEN 325  BEFORE                                   118
         NO NOT AS I CAN REMEMBER                                  119
     5 IS A FACTOR OF 325 WHICH LEAVES 65                          120
         HAVE I SEEN 65  BEFORE                                    121
         NO NOT AS I CAN REMEMBER                                  122
     5 IS A FACTOR OF 65 WHICH LEAVES 13                           123
         HAVE I SEEN 13  BEFORE                                    124
         NO NOT AS I CAN REMEMBER                                  125
     13 IS A PRIME.                                                126
     MEMORIZE 203125 AS CONSISTING OF THE FOLLOWING FACTORS        127
5  5  5  5  5  5  13
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

THE INPUT-LIST IS LONG ENOUGH TO TRY EXP ROUTINE                   129
   FIND NUMBER OF OCCURENCES OF THE MOST FREQUENT FACTORS          130
                WE HAVE 1 OCCURENCES OF 5                          131
                WE HAVE 3 OCCURENCES OF 2                          132
                WE HAVE 6 OCCURENCES OF 3                          133
                WE HAVE 10 OCCURENCES OF 2                         134
                WE HAVE 6 OCCURENCES OF 5                          135
   THE NUMBERS ARE                                                 136
1  3  6  10  6
   TAKE THE FIRST DIFFERENCE OF THE ABOVE LIST                     138
2  3  4  -4

   TRY TO FIND A LINEAR SEQUENCE EVERY ENTRY OF WHICH IS SMALLER   140
   OR EQUAL TO CORRRESPONDING ENTRY IN THE GIVEN SEQUENCE          141
   (2 EXCEPTIONS ARE ALLOWED)                                      142

          ADD 1 TO THE FIRST ENTRY OF THE INPUT-SEQUENCE. AND      143
          USE THE RESULT AS FIRST ENTRY OF THE ESTIMATED SEQUENCE
          COMPUTE AND TEST ESTIMATE                                145
              THE ESTIMATE IS NOW 2 + (X-1)* 1                     146
          OUR EST. IS LEGAL BUT IS IT THE LARGEST                  147
```

Figure 6,10.3



```
     I DO NOT KNOW MUST TRY TO INCREASE IT
     COMPUTE AND TEST ESTIMATE
          THE ESTIMATE IS NOW 3 + (X-1)* 1
     THE ESTIMATE IS TOO HIGH
          ALLOW 2 ESTIMATES TO BEE TOO HIGH (TAKES CARE OF THE
          FACTORS 0 AND 1 TOGETHER WITH 0 ERRONEOUS INPUT(S)).
     THE ESTIMATE IS TOO HIGH
          THE ESTIMATE IS TOO HIGH. REDUCE FIRST ENTRY
          AND TRY AGAIN
     COMPUTE AND TEST ESTIMATE
          THE ESTIMATE IS NOW 2 + (X-1)* 1
     OUR EST. IS LEGAL BUT IS IT THE LARGEST
     YES IT IS. WE JUST REDUCED IT BY 1 AS IT WAS TOO HIGH
THE ESTIMATED LINEAR SEQUENCE IS    2 + (X-1)* 1
MAKE A LIST OF THE EST.
2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
21
    TRY TO FIND THE BASIS OF THE EXP.
ASSUMED LIST OF BASIS
1  2  3  4  5
        NOW I HAVE AN ESTIMATE OF THE BASIS
        TEST THE EST. AND IF NECESSARY MAKE IT LINEAR
             ADD 1 TO THE FIRST ENTRY OF THE INPUT-SEQUENCE. AND
             USE THE RESULT AS FIRST ENTRY OF THE ESTIMATED SEQUENCE
             COMPUTE AND TEST ESTIMATE
                  THE ESTIMATE IS NOW 2 + (X-1)* 1
             THE ESTIMATE IS TOO HIGH
                  ALLOW 2 ESTIMATES TO BEE TOO HIGH (TAKES CARE OF THE
                  FACTORS 0 AND 1 TOGETHER WITH 0 ERRONEOUS INPUT(S)).
             THE ESTIMATE IS TOO HIGH
                  THE ESTIMATE IS TOO HIGH. REDUCE FIRST ENTRY
                  AND TRY AGAIN
             COMPUTE AND TEST ESTIMATE
                  THE ESTIMATE IS NOW 1 + (X-1)* 1
             OUR EST. IS LEGAL BUT IS IT THE LARGEST
             YES IT IS. WE JUST REDUCED IT BY 1 AS IT WAS TOO HIGH
THE ESTIMATED LINEAR SEQUENCE IS    1 + (X-1)*1
MAKE A LIST OF THE EST.
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19
20
    NOW COMPUTE THE EXP SEQ. (BASE**EXP.)
    HERE IS THE RESULT
1  8  81  1024  15625  279936
    DIVIDE INPUT BY EST. OF EXP. PART. THIS GIVES A POLYNOMIAL
    NOW TEST THE POLYNOMIAL PART
5  7  9  11  13

USE THE ALGORITHM

THE GENERAL EXPRESSION IS
(2*X + 3)

                    ESTIMATE =  5               INPUT =  5
                    ESTIMATE =  7               INPUT =  7
                    ESTIMATE =  9               INPUT =  9
                    ESTIMATE =  11              INPUT =  11
```

Figure 6.10.4

```
                         ESTIMATE =  13          INPUT =  13                         201
POLYNOMIAL SEEMS OK                                                                  202
NOW IT IS TIME TO BUILD UP THE FINAL ESTIMATE AND TO TEST IT                         203
THE ESTIMATE IS                                                                      204
5  56  729  11264  203125  4199040
THE INPUT TO THE ROUTINE WAS                                                         206
5  56  729  11264  203125
EXPONENT IS                                                                          208
2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20                   209
21
THE BASIS IS                                                                         211
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19                    212
20
THE EXPONENTIAL PART IS                                                              214
1  8  81  1024  15625  279936
THE POLYNOMIAL PART IS                                                               216
5  7  9  11  13  15  17  19  21  23  25  27  29  31  33  35  37  39
41  43
THE OUTPUT IS                                                                        219
0  5  56  729  11264  203125  4199040

THE GENERAL EXPRESSION IS                                                            221
(2*X + 3)*(1*X)**(1*X + 1)                                                           222

I WILL REMEMBER THIS                                                                 223
**********************************************************************

PRIME-FACTORS ARE NOW AVAILABLE FOR                                                  224
203125  11264  729  56  5                                                            225

TIME USED  =.836073 01  SECONDS                                                      226
```

Figure 6.10.5

6.5 An Extrapolator for Intertwined Sequences

In certain situations, sequences consisting of more than one subsequence are presented,

e.g., n_1^1 , n_2^1 .... n_i^1 , ...

and

n_1^2 , n_2^2 .... n_1^2 , ...

are intertwined as

n_1^1 , n_1^2 , n_2^1 , n_2^2 , ... , n_i^1 , n_i^2 , ...

or

n_1 , n_2 , n_3 , n_4 , ... , n_{2i-1}, n_{2i}, ...

We have, in our sequence-extrapolator, included the ability to separate two intertwined subsequences, extrapolate them separately, and recombine them. It is, of course, trivially simple to include the ability to separate any number of intertwined subsequences. We have, however, chosen not to do so. The case of two intertwined subsequences is included because it is very useful for extrapolation of oscillating sequences which occur for certain combinations of parameters of the extended polynomial sequence extrapolators.

The oscillating sequences are extrapolated in two parts, one for even and one for odd indeces.

6.6 Recognition of Previously Encountered Sequences

Unless the extrapolation procedure is extremely efficient, there should be a facility for recognition of previously encountered sequences, such that simple retrieval from memory substitutes extrapolation. SEP includes several schemes for dynamic storage of intermediate results; one of these is designed as an associative file for sequences.

The simplest case of information retrieval is the situation where a unique label can be assigned as index for storage as well as retrieval of information. The task-environment of our machine is such that it is not feasible to assign a unique label to every conceivable sequence, and even if it were, there would be no way of finding the correct label for an input-sequence without actually extrapolating and analyzing it. The first few entries of a sequence could possible be used as an identifier but, as we want our machine to recognize sub-sequences of previously encountered and extrapolated sequences, this indicator would obviously not suffice.

The simple scheme described below has been chosen for implementation.

Description of Associative Storage

1. Any entry of a sequence may be used as a clue C_i for its retrieval.
2. Any clue C_i may occur in several sequences. For efficiency in retrieval, we therefore allow the use of multiple clues. Assume m clues are used, e.g., $C_1, C_2, \ldots, C_m$.
3. The total number of conceivable clues is very large. We, therefore, for certain operations combine clues into classes K_j.[16]

4. For each class of clues K_j , there is a subset L_j of clues which have been encountered,

   i.e., if $C_i \in L_j$ then $L_j \subset K_j$.

5. For each $C_i \in L_j$, there is associated a set N_i of the names of all sequences Y_e on which C_i is known to occur,

   i.e., if $C_i \in L_j$, then $N_i = \{Y_e \mid C_i \in Y_e\}$.

The indirect definition of sets make the description somewhat awkward; an example may clarify the situation.

Given a sequence Y, (Y = 1, 3, 5, 20, 30, 40, 80, 155,) establish if it has been previously encountered. Allow for one error in the input-sequence.

1. Extract clues, by any method, in a number determined by experience.

$$C = 5, 30, 80, 155 .$$

2. Establish the class-belonging of the clues

$5 \in K_1$ i.e., $0 < 5 < 25$

$26 < 30 < 75$

$76 < 80 < 200$

$76 < 155 < 200$.

3. Retrieve L_1 , L_2 , and L_3 (corresponding to K_1 , K_2, and K_3) ,

   L_1: (1, 4, $\underline{5}$, 12, 16, 21)

   L_2: (27, $\underline{30}$, 42, 49)

   L_3: (54, $\underline{80}$, 94, 120, 155, 305)

   $5 \in L_1$, $30 \in L_2$, $80 \in L_3$, and $155 \in L_3$.

It is now established that all clues have been encountered previously. The members of L_1, L_2, and L_3 do not only represent clues, i.e., numerical values, they are also names of sets, which can be retrieved. Denote the name of 5 by N_5, etc.

4. Retrieve the sets named by clues

   N_5 : (S_4, S_7, S_9, S_{11})

   N_{30} : (S_1, S_3, S_7, S_{11})

   N_{80} : $(S_2, S_7, S_{11}, S_{14}. S_{15})$

   N_{155}: (S_7, S_{16}) .

5. Find the sequences whose names are members of at least $m - E$, (i.e., three) sets N_i . $i = 5, 30, 80, 155$.

   By inspection, we find that:

   S_7 is a member of 4 sets, and

   S_{11} is a member of 3 sets.

The two sequences, S_7 and S_{11}, are now retrieved and tested against the input-sequence. As S_7 occurs on more lists than S_{11}, it is tested first.

Input: (- - 5, 30, 80, 155, ...)

S_7 : (1, 3, 5, 30, 80, 155, 268 ....)

Storage of information in the above associative store follows essentially the same steps as outlined for the retrieval procedure.

The main differences are that clues are added to the sets L_j and that the name of the sequence containing the clues is added to the proper sets N_i . If no such sets exist, they are created.

The method for information retrieval, illustrated above, has the disadvantage that information has to be duplicated and stored in two different places, namely, as clues and as members of sequences. The advantages are that no direct addressing is necessary at any stage of the procedure, all sets are completely flexible, and any amount of information can be accommodated without modification of the scheme.[17]

The rigidity of classification may turn out to be disadvantageous, in which case an EPAM-like structure, which grows its own categories as found necessary, is contemplated for implementation.

FOOTNOTES FOR SECTION 6

1. By the definition of Δ^k and the distributive law for Δ. For a proof, see Kuntz's "Numerical Analysis", McGraw Hill, 1957.

2. Proofs of this theorem may be found in most textbooks on Algebra.

3. For a proof, see reference in Footnote 1, above.

4. All cases are normalized by appropriate translation of the coordinate-system.

5. The sequence Y^n has only one entry.

6. By permitting any order of polynomials as valid extrapolations, any sequence of numbers can be extrapolated.

7. In the presently implemented extrapolator, the values of k, l, and m, are limited to respectively, 4, 1, and 1.

8. There are a few exceptions to result 1:

   a. If $a_i = 0$ then $y_i = 0$ regardless of c_i.

   b. If $b_i = 0$ then $y_i = 0$ regardless of c_i.

   c. If $b_i = 1$ then $y_i = a_i$ regardless of c_i.

9. The value of this factor is at this stage irrelevant.

10. Note previously listed exceptions.

11. The value 2 represents exceptions a, b, and c of result 1. If three exceptions occur simultaneously, a and b will coincide.

12. With the following exceptions:

    a. If $a_i = 0$, then the value of b_i is irrelevant.

    b. The rule does not apply for erroneous y_i.

13. Except for a number of values corresponding to E.

14. The procedure amounts to guessing one of the five roots of a polynomial equation of degree 5.

15. k + 1 entries are needed for the polynomial part, however, the exponential part may use the same entries and it often needs less than k + 1 entries for identification.

16. Such classification greatly reduces the time-requirement for searching the storage.

17. As long as the storage is not exhausted.

BLANK PAGE

7. Extrapolation of Sequencies of Symbols

Models based on the extrapolation for the Thurstone-letter-completion tests, described in Section 4, are presented in this section; generalizations as well as particular implementations are discussed.

7.1. A Basic Extrapolator

The present extrapolator is included in SEP. It is based on the graphical representation of test-sequencies given in Section 4.2.2. The computer-program is written in the programming-language IPL-V, and certain features of this language has dictated the design.

Let us demonstrate the extrapolating procedure in terms of an example. Find the next entry of the sequence:

w x a x y b y z c z a d a b _

1. Reverse the order of the sequence.[1]

_ b a d a z c z y b y x a x w

2. Identify the periodicity of the sequence. "Guess" the most likely periodicity k, assume $k = 3$. If the guess is correct, the underlined symbols should form a "legal" sub-sequence.

_ b a $\underline{d}$ a z $\underline{c}$ z y $\underline{b}$ y x $\underline{a}$ x w

There are only three legal sequencies. They are partially represented below.

d	d	d	d	d	(=)
c	d	e	f	g	(+)
e	d	c	b	a	(-)
-	d	c	b	a	(assumed sub-sequence)

3. Comparison of sequencies gives the result that the next letter is e .

It should be observed that the here outlined procedure is based exclusively on ordinal properties of the alphabet, nd thus does not require any numerical computations. We want to stress this fact because certain sequence-extrapolators employ a policy like: "Any symbol can be encoded into numbers, therefore, a numerical extrapolator can be employed for any sequence of symbols."[2] The risk of using this policy is that the ordinal character of alphabets is replaced by the cardinal properties of numbers. So, for instance, the sequence:

a d i p or encoded 1 4 9 16

could be extrapolated:[3]

1 4 9 16 25 which is decoded a d i p y .

An important advantage of using purely symbolic comparisons is that any alphabet or actually any number of different alphabets can be introduced as data for the sequence-extrapolator. We, thus, in our case usually include at least the English alphabet and the alphabet of vowels.[4] When several alphabets are employed, they are ranked according to the relative frequencies of their occurrence in input-sequences. Input-sequences are tested against one alphabet at a time until success or failure of the extrapolation is established. A print out of results derived by the present extrapolator is given in figure 7.1.

7.1.1 A Numerical Version of the Basic Extrapolator

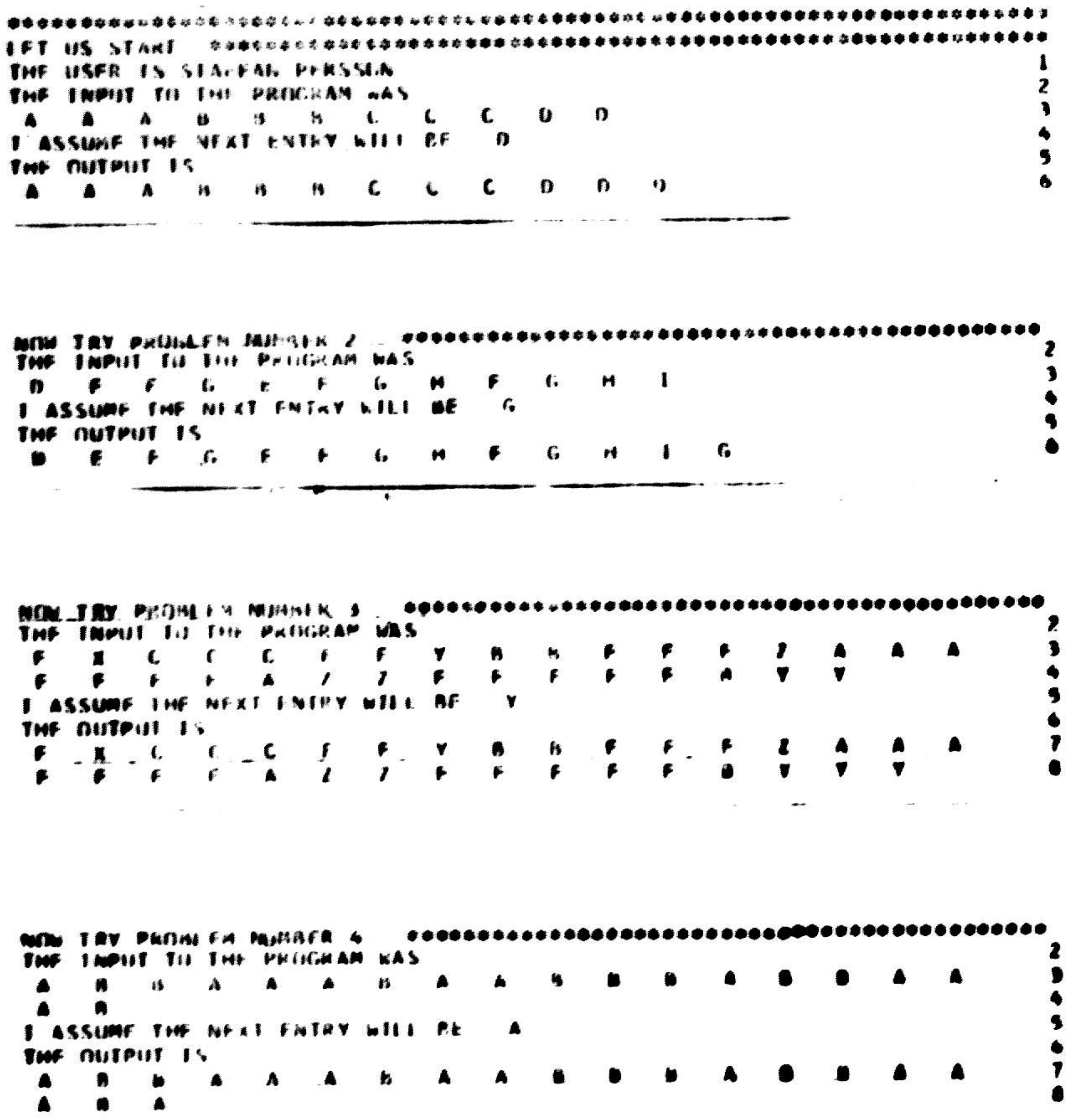
We can easily extend the domain of our basic extrapolator to cover arbitrary, but constant, step-sizes for each sub-sequence. The most convenient way of implementing this extension is to use numerical encoding for the input-sequence (N.B.: the warning concerning this approach given in the preceding section). By restricting the domain of the numerical extrapolator to linear sequencies most objections against the approach may be avoided.

Procedure:

1. Encode the alphabet into numbers.

a = 1 (or 27)
b = 2 (or 28)
. .
. .
. .
z = 26 (or 52)

2. Instead of trying to match the encoded sequence against a set



```
****************************************************************************
LET US START   *************************************************************
THE USER IS STAFFAN PERSSON
THE INPUT TO THE PROGRAM WAS
 A   A   A   B   B   B   C   C   C   D   D
I ASSUME THE NEXT ENTRY WILL BE   D
THE OUTPUT IS
 A   A   A   B   B   B   C   C   C   D   D   D



NOW TRY PROBLEM NUMBER 2   **********************************************
THE INPUT TO THE PROGRAM WAS
 D   E   F   G   E   F   G   H   F   G   H   I
I ASSUME THE NEXT ENTRY WILL BE   G
THE OUTPUT IS
 D   E   F   G   E   F   G   H   F   G   H   I   G



NOW TRY PROBLEM NUMBER 3   **********************************************
THE INPUT TO THE PROGRAM WAS
 F   X   C   C   C   F   F   Y   B   B   F   F   F   Z   A   A   A
 F   F   F   F   A   Z   Z   F   F   F   F   F   A   Y   Y
I ASSUME THE NEXT ENTRY WILL BE   Y
THE OUTPUT IS
 F   X   C   C   C   F   F   Y   B   B   F   F   F   Z   A   A   A
 F   F   F   F   A   Z   Z   F   F   F   F   F   A   Y   Y   Y



NOW TRY PROBLEM NUMBER 4   **********************************************
THE INPUT TO THE PROGRAM WAS
 A   B   B   A   A   A   B   A   A   B   B   B   A   B   B   A   A
 A   B
I ASSUME THE NEXT ENTRY WILL BE   A
THE OUTPUT IS
 A   B   B   A   A   A   B   A   A   B   B   B   A   B   B   A   A
 A   B   A
```

Figure 7.1

of "legal" sequences, we now try to find linear sub-sequences of the form:

$C_i = C_k + \Delta \cdot i$ where: C_i = the i:th member of the sub-sequence.

Δ = a constant (first difference).

3. If a legal subsequence containing the blank _ is found, the value corresponding to _ is computed.

3x Under certain circumstances, several "legal" subsequences of different periodicity may be found; in this case, the smallest value k for the periodicity is chosen.

EXAMPLE: Find the next entry of the sequence:

	a	x	d	y	g	z	j	a	_
Encode:	1	24	4	25	7	26	10	1 (27)	_

Assume that the periodicity is = 2.

1 4 7 10 _

Is this subsequence linear?

From the two first entries, a linear sequence is generated:

$$C_i = 1 + (4 - 1)(i - 1) \quad \text{or} \quad C_i = -2 + 3i \ .$$

We generate for $i = 1, 2, 3, \ldots.$

1 4 7 10 13 ...

Now, tect the generated sequence against the given subsequence. As the two sequences are identical in the first four positions, the value $C_5 = 13$ is accepted for _ .

We now have:

1 4 7 10 13

Or, for the "complete" sequence:

1 24 4 25 7 26 10 27 13

Decode:

a x d y g z j a m

7.2. A "Two-Dimensional" Extension of the Basic Extrapolator

a b b c c c d d d _

The continuation of the above sequence seems obvious. We have, therefore, decided to extend the problem-domain of the basic extrapolator to include a class of "two-dimensional" sequences, which, as a simple special case, can solve the above problem.

The procedure is based upon re-representation of "runs" of symbols by a pair (s , n) where s is the symbol and n is the number of occurrences of s .

EXAMPLE:

b b g w w b b b h v v b b b b i u u b b b b b j t t _ is

represented as

(b,2)(g,1)(w,2)(b,3)(h,1)(v,2)(b,4)(i,1)(m,2)(b,5)(j,1)(t,2)

or, as no confusion can result, as

b 2 g 1 w 2 b 3 h 1 v 2 b 4 i 1 m 2 b 5 j 1 t 2 _ .

which, in the model of section 4, is represented as

(b, 2, g, 1, w, 2) , (= , + , + , = , - , =) .

This model can be applied to generate the continuation b

The present extrapolator can be used to extrapolate most of the numerical examples of section 4, e.g., the sequence 1, 11, 111, 1111, 11111, is represented by the sequence

$$\underbrace{11}_{1}\ \underbrace{,1}_{,}\ \underbrace{12}_{11}\ \underbrace{,1}_{,}\ 13\ ,1\ 14\ ,1\ 15\ ,1$$

etc.

which is translated into the model

(1, 1, , , 1,) (=, +, =, =) .

which is extrapolated by generation of as many entries as necessary to produce a comma-sign, e.g.,

11, 112, 113, 114, 115, 116,

Retranslation to a "one-dimensional" sequence gives:

1, 11, 111, 1111, 11111, 111111 .

Sample-outputs from the extended extrapolators are given in Figure 7.1.

7.2.1 A Modified "Two-dimensional" Extrapolator

The following sequence cannot be extrapolated by the method of section 7.2 because some adjacent members of the subsequencies are identical.

c e e d d d d d d e c c d f b b b d g a a d h _ .

However, by utilizing the restrictions of the model of section 4, a revised procedure can be employed for its extrapolation.

1. Translate into a "two-dimensional" sequence

   $\underline{c}$ 1 e 2 $\underline{d}$ 6 $\underline{e}$ 1 c 2 d 1 $\underline{f}$ 1 b 3 d 1 $\underline{g}$ 1 a 2 d 1 $\underline{h}$ 1 _ .

2. Try to find a legal subsequence, regardless of periodicity

   c d e f g h .

3. Retrieve the quantities corresponding to the subsequence

c 1 d 6 e 1 j 1 g 1 h .

The resulting sequence should be valid in the model of Section 4.2.1. If it is not so, make the necessary adjustments:

(c , 1)(+ , =) → c 1 d 1 e 1 j 1 g 1 h 1 .

4. Subtract the "candidate" subsequence from the initial "two-dimensional" sequence. The remainder is:

e 2 d 5 c 2 d 1 b 3 d 1 a 2 d 1 .

5. Repeat steps 2-4:

2. e d c b a

3. e 2 d 5 c 2 b 3 a 2

(e, (2, 3)) , (- , (= , =)) → e 2 d 3 c 2 b 3 a 2

4. d 2 d 1 d 1 d 1

5. d d d d

2. d 2 d 1 d 1 d 1

3. d 2 d 1 d 1 d 1 → d d d d d → d 1 d 1 d 1 d 1 d 1

4. ∅ .

6. Assemble the subsequences:

```
C1          D1          E1          F1          G1          H1
     E2          D3          C2          B3          A2          _
        D1          D1          D1          D1          D1
_________________________________________________________________
C1  E2  D1  D1  D3  D1  E1  C2  D1  F1  B3  D1  G1  A2  D1  H1  _ .
```

7. Employ the model M to find continuation

(c, 1, e, (2 , 3), d, 1)(+, =, =, (= , =), = , =) .

8. The result is z .

7.3 An Extrapolator for Non-regular Sequences

The present extrapolator utilizes certain statistical properties of a sequence to "guess" its most likely continuation.

A short digression may clarify the bases of its design.

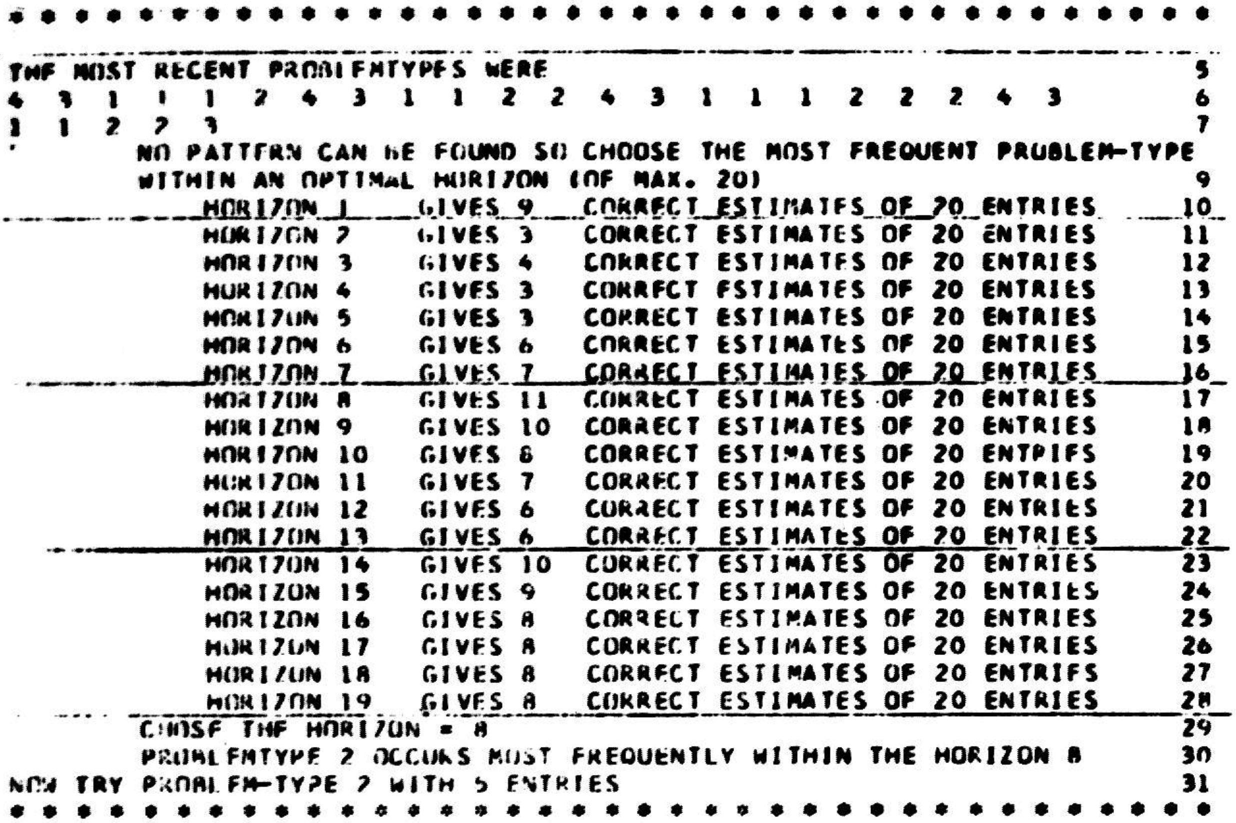
Given an infinite sequence, generated by a good pseudo-random-generator, then the best strategy of predicting its next member is to produce the, in the past, most frequently occurring number. Or, in other words, the most frequently occurring member under an infinite horizon.

Given a sequence where members occur in randomly distributed runs of random length, then the best strategy would be to always select the latest occurring member as prediction of the next occurrence.[5] Or, in other words, the most frequently occurring member under a horizon of 1.

Given sequences of other designs, it is likely that a strategy such as selecting the most frequently occurring member under an optimally long horizon will prove suitable.

The present sequence-extrapolator tests the given historic sequence, $y_2\ y_1\ \cdots\ y_n$, for <u>all</u> horizons h from 1 to n and chooses the horizon which provides the highest number of correct predictions on the n first items. Because of the finite length of the sequence, predictions on the first few items $y_{,}\ \ldots\ y_n$ will be based upon a horizon smaller than h.

An example of a computer-output from the extrapolator is given in Figure 7.2.



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THE MOST RECENT PROBLEMTYPES WERE
4  3  1  1  1  2  4  3  1  1  2  2  4  3  1  1  1  2  2  2  4  3
1  1  2  2  3
.
         NO PATTERN CAN BE FOUND SO CHOOSE THE MOST FREQUENT PROBLEM-TYPE
         WITHIN AN OPTIMAL HORIZON (OF MAX. 20)
              HORIZON 1     GIVES 9    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 2     GIVES 3    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 3     GIVES 4    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 4     GIVES 3    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 5     GIVES 3    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 6     GIVES 6    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 7     GIVES 7    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 8     GIVES 11   CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 9     GIVES 10   CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 10    GIVES 8    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 11    GIVES 7    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 12    GIVES 6    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 13    GIVES 6    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 14    GIVES 10   CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 15    GIVES 9    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 16    GIVES 8    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 17    GIVES 8    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 18    GIVES 8    CORRECT ESTIMATES OF 20 ENTRIES
              HORIZON 19    GIVES 8    CORRECT ESTIMATES OF 20 ENTRIES
         CHOSE THE HORIZON = 8
         PROBLEMTYPE 2 OCCURS MOST FREQUENTLY WITHIN THE HORIZON 8
NOW TRY PROBLEM-TYPE 2 WITH 5 ENTRIES
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Figure 7.2

FOOTNOTES FOR SECTION 7

1. This is done for convenience of programming; it is not a necessary step.

2. Pivar and Finkelstein (Section 6, ref. [3]) use this approach; they, however, mainly discuss "linear" sequences but also claim, "...compounding of operations is necessary when dealing with more complex sequencies."

3. The sequence is assumed to be X .

4. i.e , AABECIDOE_ is extrapolated
AABECIDOEU .

5. Compare the concept "frequency of diagrams" in cryptology.

BLANK PAGE

8. Summary

The present section, after a brief summary of previous sections, demonstrates how features of epistemological models of inquiry are implemented in the design of our sequence-extrapolating system (SEP), and furthermore, indicates how experience gained in the design of SEP can be generalized to apply in a wide range of technological models of inquiry.

Section 1. introduces the purpose of this thesis: an investigation into the feasibility of designing mechanized inquiring-systems for finding suitable representations of problems, i.e.; to perform the "creative" task of finding analogies. Because at present a general solution to this problem does not seem to be within reach, the feasibility of mechanizing a particular representational inquirer is chosen as a reasonable first step towards an increased understanding of the general problem. It is indicated that by actually designing, programming, and running a representational inquirer as a program for a digital computer, a severe test of its consistency and potential for future extensions can be performed. A short discussion of the use of models for analysis of complex "real" problems is also given.

Section 2. reviews several proposed systems of inquiry in order to indicate the possibilities of performing investigation by systems where no technical limitations exist. Although our goal is to translate the "unlimited" epistemological systems of inquiry presented by the philosophers into mechanical design, i.e.; to reduce the epistemological problems to a technical level, the discussion clearly indicates how the representational problem (in the form of a priori knowledge) assumes an increasingly important role in more sophisticated systems. A convenient classification of inquiring-systems, i.e., in Leibnitzian, Lockean, and Kantian systems is employed.

Section 3. discusses the problem of representation raised in Section 2. As no general results are available, the presentation is given in terms of specific examples. However, certain generally arising problems of representation are identified, (and exemplified in the discussion of the domino problem).

Section 4. digresses into the area of sequence-extrapolation. The discussion of context in this section, however, is analogous to the representational aspects presented in Sectiᴐns 2 and 3. This section links the general discussion of inquiry to the particular problem of mechanized sequence-extrapolation.

Section 5. is based on a description of a particular computer-program, SEP. However, the discussion is performed at such a level that the executive structure of SEP can be directly translated into a far more general class of inquirers, in fact, the executive functions are basically context-independent.

Sections 6 and 7. present particular strongly context-dependent programs for sequence-extrapolation. The individual programs are Leibnitzian but correspond to Kantian maximal <u>a priori</u> sciences, i.e.; to alternate representations of symbolic data. The presentation is given at a level which permits replication of the logic of the programs, however, no detailed discussion at the actual programming level is given.

The summary clearly indicates how our design of a representational inquiring system for sequence extrapolation has been derived from the general epistemological models of inquiry. However, it remains to be shown how experience gained in the design of SEP can be generalized to a wider area of applicability.

The limited availability of resources makes a strict translation of epistemological models of inquiry into technological designs difficult. This means that several functions can only be approximately implemented. Furthermore, the concept of efficiency will have a dominating influence on particular designs. Still, certain major features of the epistemological models of inquiry are directly reflected in the design of SEP, such as:

1. the Leibnitzian logical processor (Section 2.1);
2. the Kantian employment of *a priori* sciences (Section 2.3);
3. the Lockean capability of reflection on the internal processing (Section 2 2); and
4. the Singerian idea of bringing the problem-concocter into the domain of inquiry.

A detailed discussion of these features follows:

1. SEP is organized around a set of *Leibnitzian* processors which, although complicated in themselves, do not suggest any general rules of design. Nevertheless, their organization and modes of communication, i.e., the executive and the organizational functions, are most interesting from the general aspect of efficiency.

2. The multiple *a priori* sciences of Churchman's version of Kantian inquiring systems (Section 2.3) has been implemented in SEP; however, it is not quite obvious if a given technological model is to be classified as Leibnitzian or Kantian In the case of SEP, the complete system

viewed as a translation from the input to the output is Leibnitzian, and furthermore, each sub-machine (including the executive) is also Leibnitzian. Thus, SEP is, in fact, a two-stage Leibnitzian inquiring-system operating under a "time-conscious" executive. This, of course, is true of all mechanized inquirers. However, SEP approximates a Kantian design, which is reflected by the fact that its level of sophistication permits greater efficiency in terms of available resources than a purely Leibnitzian design. Moreover, the multiple _a priori_ sciences permit classification of problems, such that suitable representations may be found.

We have shown (Section 4) that not only the efficiency, but also the quality of problem solving depends upon the partitioning of a wide domain of potential representations into suitable domains for the _a priori_ sciences, but no rules for such partitioning have been given. Such rules can, however, be deduced from the functions of the executive in a Lockean design.

3. The Lockean capability of reflection on the internal processing of the inquiring system is implemented in the executive of SEP. Implementation of a vague, or intuitive rule, such as "the operations of cur own minds within, as the objects of reflection," (Section 2.2) requires a formal definition of introspective reflection. In the case of SEP, the reflections of the "inquiring executive" (Section 5.5) are analogous to processing of the "external material things as subjects of sensations" (Section 2.2). That is, the executive operates upon internally generated strings of symbols, hence, the problem arises of where, and how these strings are generated. In SEP observation can be performed on the communication between submachines and/or between submachines and the

executive. This suggests that the composition of the a priori must not only be considered as a sufficiently large set of sub-machines to "cover" a specified domain of inquiry, but that the relative domains are also important.

4. It has been shown (Section 5.5) that the division into sub-domain of SEP should be homomorph to the problem-concocters conception of classes of problems. Thus, flexibility requires a great variety of classes, i.e.; a large number of a priori sciences.

9. Conclusions

This thesis ~~has~~ attempted pursued a limited goal; to design a system for finding suitable representations in a specific environment. This goal has was ~~been~~ achieved in that a model, which meets reasonable requirements of performance for such a system, ~~has been~~ programmed and tested on a digital computer.

In the light of the discussion of Section 8, it may be argued that the ~~our~~ two-stage Leibnitzian inquiring system represents an algorithm for finding algorithms, however, this is (as shown ~~in Section~~ 3) always true of the representational mode of information. Therefore, such arguments must be based on a more restricted conception of the representational problem in mechanized inquiry than is possible in the scope of this dissertation.

The presented model permits generalization in several directions due to the basically context-independent design of the executive structure. Although it does not represent the only, and hardly even the best, way to design the executive structure of a complex representational inquiring-system, it is felt that a direction for future research is to be found along the lines of reasoning presented here.