MEMORANDUM
RM-5085-PR
SEPTEMBER 1966

A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL

R. E. Levien and M. E. Maron

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-5085-PR
SEPTEMBER 1966

A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL

R. E. Levien and M. E. Maron

This research is sponsored by the United States Air Force under Project RAND—Contract No. AF 49(638)-1700—monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

PREFACE

This Memorandum describes the objectives and approach of a current RAND project concerned with the use of computers as assistants in the logical analysis of large collections of factual data. The two principal tasks are the development of techniques for storage and retrieval of millions of data sentences and provision of a programming system in which the processes of logical analysis may be expressed. Using those tools the analyst will be able conveniently to gain access to and draw inferences from data files too large for extensive manual investigation. To ensure their practicality, the techniques are being tested on a large corpus of factual information concerning research in cybernetics.

The Memorandum is intended to provide a brief and informal description of the research project for those engaged in related research and for potential users. Its general view of the entire project provides the background for more detailed technical descriptions of specific portions of the project that are to appear in forthcoming memoranda.

The research reported here should be of interest to Air Force organizations concerned with storage, retrieval, and inference-making from large files of data. Among the potential areas of application are command and control, research management, technical information dissemination, and advanced development planning.

SUMMARY

A potentially important computer application area is the logical analysis of large collections of factual data. Its implementation requires development of techniques for storage and retrieval of millions of data sentences and provision of a programming system in which the processes of logical analysis may be expressed.

Among the specific problems that must be solved are logical and linguistic problems of representing the basic data, file problems of organizing the data for rapid and direct access, programming problems of designing a language for convenient interrogation of such a file, and hardware problems of providing sufficiently large, rapid access storage.

In the Relational Data File under development at The RAND Corporation, data is represented by sentences in an artificial information language. Each sentence expresses a binary relationship between two entities. The sentences are directly accessible by content. The file of these sentences is organized in quadruplicate in a disk memory so that access time is minimized. A programming language has been designed that enables a user of the system conveniently to express not only direct retrieval requests, but also the inferential processes that are required to derive implied conclusions.

The system has been designed for use on an IBM 7044 with an IBM 1301 disk. The system is partially operational and is expected to be fully operational late in 1966.

CONTENTS

ACKNOWLEDGMENTS

The early ideas and plans for the research described in this paper were developed by the two authors, but we have been fortunate to have gained competent colleagues who are participating in what is now a joint effort. We are deeply indebted to Don Cohen and Gerald Levitt for their excellent execution of the enormous programming tasks. The logical and linguistic problems at the core of the research are complex, and we have been fortunate to have the constant and competent collaboration of J. L. Kuhns. Moreover, we are grateful for the diligence of Wade Holland and his able assistants in handling the severe input problems.

I. INTRODUCTION

The computer's potential value as an assistant in the logical analysis of large collections of factual data was clearly and explicitly recognized soon after construction of the first automatic sequence-controlled calculator. Vannevar Bush wrote in his famous Atlantic Monthly article:(1)

> The repetitive processes of thought are not confined ...to matters of arithmetic and statistics. In fact, every time one combines and records facts in accordance with established logical processes, the creative aspect of thinking is concerned only with the selection of the data and the process to be employed and the manipulation thereafter is repetitive in nature and hence a fit matter to be relegated to the machines.
>
> ...
>
> The scientist...is not the only person who manipulates data and examines the world about him by the use of logical processes... . Whenever logical processes of thought are employed--that is, whenever thought for a time runs along an accepted groove--there is an opportunity for the machine.

Such opportunities for the machine are myriad. Two will serve to illustrate the promise.

A corporate data analysis system would contain a computer store of factual data about the products, markets, facilities, finance, plans, and personnel of the firm. Corporate managers would employ it to answer questions of fact and as an assistant in estimating consequences, testing alternatives, and drawing inferences.

A scientific activities data exchange would comprise a large store of factual data about the scientists, projects, publications, organizations, conferences, and contracts associated with research in a particular field of science. To scientists, administrators, and editors

it would be a source of data and an assistant in drawing conclusions about relevant publications, interested researchers, competent organizations, and important trends. Unlike literature searching systems, it would also enable specific information about persons, organizations, and projects to be retrieved through searches of a wide variety of factual data about the scientific context.

The promise is there. What of the progress? In the twenty or so years since Bush wrote his article, computers have frequently been used to store, select, and manipulate many kinds of data. And in the last decade efforts have been made to mechanize the execution of logical analysis as applied to theorem proving and problem solving. Nevertheless, the application that Bush envisaged is yet to be realized. The combination of a large data base and the tools of logical analysis in a single system has not been provided. In fact, it has been tried only rarely.

In 1962 Manfred Kochen proposed the AMNIPS system,[2] which did combine storage of a large body of factual data with the ability to perform logical analyses of that data. A project to implement AMNIPS was under way at IBM until early 1965,[3] but is no longer active. Related proposals (some supported by experimental programs) have been made by Black, Cooper, Elliott, Kugel, Lindsay, McCarthy, Raphael, Slagle, Thompson, and Travis.[4-14] The work on "question-answering" systems has explored the same area, although usually the emphasis in such studies has been on translating natural language questions into a form that can be answered by direct retrieval from a file. Simmons' article[15] provides an excellent survey of work in that field.

However, none of these research projects has yet culminated in a system that would be a practical assistant in the logical analysis of large bodies of data.

In 1963 design and development of the Relational Data File began at The RAND Corporation. It is a system intended to serve the needs described above.[16] An initial version should be operational by the end of 1966. This Memorandum describes the problems that arise in the course of implementing such a logical analysis system and indicates how they have been resolved in the Relational Data File.

II. REQUIREMENTS

A computer system that will assist in the logical analysis of data must possess two principal features:

(1) the capacity to store a large body of factual data;

(2) the ability to execute logical analyses of the data.

Let us examine each of these requirements in more detail.

DATA BASE

The primary question is: What characteristics must a data base have if it is to be part of a computer system that will assist in logical analysis?

First, if there is to be benefit from using the computer and if the analyses are to be nontrivial, the data base must be extremely large--at least, on the order of 10^5 or 10^6 data statements.

Second, if logical analysis is to be efficient, the data base should be organized as a single-level pool of elementary statements. The conventional organization of data bases into files, which contain records, which in turn contain fields, is too awkward for efficient logical analysis, in which the smallest unit of information must be directly accessible for computer manipulation.

Third, since most logical analyses will require many file accesses and searches for data concerning specific objects, elementary data should reside on random-access storage media and be accessible on the basis of content. All data about a particular individual, for example, should be retrievable without recourse to a sequential file search.

Thus, the data base should contain 10^5 or 10^6 elementary data items arranged in a single-level pool for random access on the basis of content. In Sec. III we consider the implementation of such a data base for the Relational Data File.

LOGICAL ANALYSIS

The second major question is: What is required in order to execute logical analyses of the data base?

We should observe, first, that information about the 10^5 or 10^6 factual data explicitly represented in the data base would confer a degree of plausibility on many other factual data, not themselves explicitly stored. Those implicit data may be obtained from the explicit data through the process of inference.

One traditional form of an inference is "If P, then C," where P, the premise, is some possibly complex logical condition, and C, the conclusion, is the statement whose truth or plausibility is conferred by the truth of P. Two types of inference need to be distinguished: strict (or deductive) and plausible (or inductive). In strict inference, the truth of the premise implies the truth of the conclusion; in plausible inference, it only confers some degree of plausibility (less than truth) on the conclusion.

A computer assistant for logical analyses must aid in the process of inference. That process has two phases: inference specification, deciding what logical conditions must be satisfied in order to assert a desired conclusion; and inference execution, searching the data base for combinations of elementary data that satisfy the premise and asserting the appropriate conclusion. These phases are analogous to

algorithm specification and algorithm execution. And inference specification, like algorithm specification, is best done by man. But inference execution, like algorithm execution, is an appropriate job for the computer. Thus, the primary role for a computer assistant in the logical analysis of large data bases is inference execution.

In order for a computer to execute complex inferences, it must be able to execute the appropriate set of more elementary logical operations, and a language must exist in which the user can specify the inferences he wishes to have executed. These two requirements may be subsumed under the single problem of designing a programming language and its translator.

Thus, in order to implement logical analyses of the data base, it is necessary to provide a programming language for inference specification and execution. In Sec. IV we consider the design of such a language for the Relational Data File.

III. STRUCTURE OF THE DATA FILE

Designing data bases that will be amenable to logical analysis is a task that must proceed on several levels. At the highest level is the design of a language in which to express the relevant events, situations, and qualities. It is a logical and linguistic question, essentially independent of the eventual computer implementation. The next level is the design of the internal representation of each linguistic expression. Here, the properties of the computer must be taken into account. Finally, there is the organization of the individual linguistic expressions within the total file. At that point, the probable output demands must be a controlling design consideration. In addition, there is the problem of collecting and entering the data. In this section, those problems of data storage and retrieval will be discussed. The design of a file of data for a scientific activities data exchange will provide specific examples.

LOGICAL AND LINGUISTIC PROBLEMS

What type of language should be used?

Ordinary language has much richness, flexibility, and versatility; however, it suffers from its lack of precision. Sentences in ordinary language can be ambiguous; several different sentences can be synonymous; meaning depends on context. Computers cannot now be programmed effectively to resolve the imprecision, synonymy, or ambiguity of ordinary language for purposes of logical analysis. Hence, ordinary language is unsuitable for the data base.

Artificial languages can be designed to be precise, unique, and unambiguous. Following Uspenskii,[17] we call a declarative artificial language an information language.

Since we are concerned with the logical analysis of factual data, it is natural to look to formal logic for the structure of an information language. In the RAND Relational Data File we use that part of modern logic called the "predicate calculus." It permits description in terms of properties and relationships among individuals. To illustrate that viewpoint, consider the following three sentences in ordinary language:

1. A. P. Smith is a system engineer.
2. A. P. Smith is affiliated with Acme Electronics Corporation.
3. A. P. Smith received the Ph.D. degree from UCLA.

The first sentence expresses a property--that of being a system engineer--of an individual, A. P. Smith. In the predicate calculus we might symbolize that situation by:

A. P. Smith/SYSTEM ENGINEER/

where "SYSTEM ENGINEER" is the name of a carefully defined property.

The second sentence expresses a relationship--that of being affiliated with--between two individuals, A. P. Smith and Acme Electronics Corporation. Notice that we use the term "individual" to mean "entity" and not simply "person." Sentence 2 might be symbolized by:

A. P. Smith/AFFILIATED WITH/Acme Electronics Corporation

where "AFFILIATED WITH" is the name of a precise binary relation.

The third sentence expresses a relationship--that of receiving the degree of ... from--among three individuals, A. P. Smith, Ph.D., and UCLA. It could be symbolized by:

A. P. Smith/RECEIVED/Ph.D./DEGREE FROM/UCLA

where "RECEIVED ... DEGREE FROM" is the name of a precise ternary relation.

Names of properties are also called "one-place predicates;" binary relations, "two-place predicates;" and ternary relations, "three-place predicates." There are, of course, increasingly complex relationships that can be expressed formally in terms of many-place predicates. The design of a formal information language based on the predicate calculus, therefore, reduces to the specification of a set of predicates in terms of which all the relevant states-of-affairs will be expressed.

In addition to the predicates given above, an information language for the context of computer science research would include:

"AUTHOR OF"

"PROGRAMMED COMPUTER"

"LOCATED IN"

"CITED IN"

"STUDENT OF"

and many others.

For simplicity of file structure--and not for logical or linguistic reasons--we translate all one-place and three- or more-place predicates into two-place predicates, i.e., names of binary relations. There are a number of ways to do so. Sentence 1 above may be expressed in terms of a binary relation "HAS PROFESSIONAL SPECIALTY":

A. P. Smith/HAS PROFESSIONAL SPECIALTY/system engineer

Many-place predicates are more difficult. We use relational composition, in which a sentence may be an entry in another sentence. Sentence 3 above would become:

A. P. Smith/RECEIVED DEGREE/(Ph.D. AWARDED BY UCLA)

However, instead of writing out a sentence each time it is composed with another, we assign names to the sentences. The above composition would be represented by two sentences, S1 and S2, as follows:

S1: A. P. Smith/RECEIVED DEGREE/S2

S2: Ph.D./AWARDED BY/UCLA

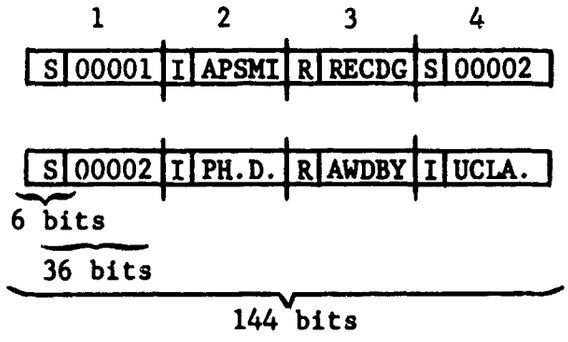
The same technique may be used to describe four-, five-, or more-place relations in terms of binary relations. Thus, we employ a relational information language. Kochen employed a similar language and suggested a technique like relational composition, which he called "nesting of sentences." Similar languages have also been suggested by Black, Elliott, Kugel, Lindsay, Raphael, and Travis.

INTERNAL REPRESENTATION

How should each information language sentence be represented inside the computer store?

Since we have limited the predicates to binary relations, each sentence has four constituents: sentence name, name of first individual (which we call the domain element), relation name, and name of second individual (which we call the range element). But both the constituents and the sentences as a whole are of highly variable length. Hence, to standardize and reduce the average length of the sentences, each constituent is encoded by a computer code dictionary at the time it enters the file. The encoded representation of a constituent is one computer word--36 bits in our implementation on an IBM 7044. The first six bits are a syntactic marker that indicates whether the constituent denotes a sentence (S), relation (R), or individual (I). (Several other possibilities exist, but are not relevant here.) The remaining 30 bits

are the assigned code. Thus, each sentence is represented by four 36-bit computer words. The two sentences, S1 and S2, shown above might become:



(In order to improve readability, the 36 bits have been translated into six alphanumeric characters.)

FILE ORGANIZATION

How should the 10^5 or 10^6 sentences be arranged in the file?

The main store of the Relational Data File is an IBM 1301 disk storage unit. Consequently, the file should be organized so as to minimize the mechanical movement of the read-write heads in executing common retrieval requests.

The most common requests will not be for a single sentence, but for all sentences that have one or more specified constituents. For example, the user may wish to know all values of x for which sentences of the form

x/AFFILIATED WITH/Acme Electronics

appear in the file. Or, he may wish all sentences of the form

A. P. Smith/R/y,

where R and y are variables. To minimize access times, such groups of sentences should be located contiguously on the disk and read as a unit into core in a single access.

The problem is in deciding how to group sentences. If, for example, sentences were ordered according to relation name in the file, access according to domain element or sentence name would demand sequential searches of the entire file. The same problem applies to files ordered according to any of the other constituents.

Our approach is to construct four files, one ordered by each constituent: sentence name, domain element, relation name, and range element. Then if all sentences whose domain element is "A. P. Smith" are sought, the domain-ordered file is used. A table in core points to that segment of the disk containing any such sentences. A replicated file obviously consumes a large amount of storage, but it allows quicker access than other alternatives. Moreover, since by appropriate organization not every constituent need be present in all four files, an efficient replicated file can be constructed using just eleven words per sentence.

Since we want to minimize access time at the expense of storage, we have chosen the replicated organization. With it--and allowing space for code dictionaries and other auxiliary data--the IBM 1301 disk (one module) will hold 232,000 sentences in the relational information language.

DATA ENTRY

Thus far we have considered how to structure and arrange sentences once they have arrived in the file. We can now turn to the question of getting them there. Since we are contemplating files that contain millions of sentences, this is a significant practical question.

Data sources fit no simple categories and take no standard form. Data on the context of scientific research appear in technical papers, research reports, proposals, bibliographic citations, survey articles, questionnaires, reference books, conference programs, and newspaper reports. In some cases--bibliographic citations, questionnaires--it is in reasonably fixed format; in others--acknowledgments, survey articles--it may be embedded in free-flowing English text. The first problem of data entry, therefore, is fitting the input data into a form convenient for machine handling. This is a task for people, and if it is to be practical, the extraction and structuring of the data for machine entry must be straightforward and precise: that is, it must be convenient for the person, as well as for the machine.

Data entry directly in the relational information language would eliminate the need for subsequent translation; however, considerable effort would be necessary to identify, record, and check the large number of sentences needed to record the data; the probability of mistakes and omissions would be high; and changes in the information language would necessitate re-extraction of the data from original sources. Data entry in the English language would be convenient for the input aides, but inconvenient for the computer.

Consequently, we have chosen to use special input forms. The input aides extract information from the original sources and enter it on the forms in a specified format. The forms are keypunched and entered on tape. Programs written in a special programming language, called FOREMAN, are then applied to the tape to derive information-language sentences that are entered into the file.

A form is designed for each class of data encountered at input. In the scientific activities data exchange, for example, forms to collect bibliographic, organizational, biographic, and project data (among others) are needed. The specific design of such forms must be done anew for each area of application, although some forms will serve several areas.

A new translation program is prepared, in the FOREMAN language, for each class of data forms. The translation program is an implementation of rules such as the following: For each author, a, whose name appears on a bibliographic form concerning publication, p, compose a sentence

a/AUTHOR OF/p

and enter it in the file.

The program is executed on each data form, producing a large set of information-language sentences for insertion in the file. If the information language changes or the data in a form are analyzed differently, a modified translation routine is prepared; once that is done, the computer easily retranslates old data forms into revised data sentences.

IV. LOGICAL ANALYSIS OF THE DATA

Perhaps the best way to describe the use of the Relational Data File as an assistant in logical analysis is through a series of examples.

Let us suppose that someone has approached the Relational Data File with an interest in A. P. Smith. Perhaps he just wishes to verify that A. P. Smith works at the Acme Electronics Corporation. Since the answer may be explicitly in the file, it is not yet a matter for inference. The user would formulate a request in the programming language as follows:

1.0. IF "A. SMITH"/"AFFILIATED WITH"/"ACME ELECTRONICS",
THEN PRINT "YES", ELSE PRINT "NOT KNOWN"

The relational sentence would be encoded by means of the code dictionary and a match for it sought in the data file. Although this is the simplest form of request, it illustrates the difficulties that arise from synonymy and ambiguity in ordinary usage.

Synonymy occurs when there are multiple names for the same object--individual or relation. In the example, the Acme Electronics Corporation has been called "Acme Electronics." The names that occur in the basic sentences of the file are selected standard names; all other synonyms should be mapped into the standard before data input or search. That function is performed by the code dictionary, which assigns the standard internal code to each synonym. In this instance, "Acme Electronics" will be assigned the same internal code as "Acme Electronics Corporation."

Ambiguity arises where one or more different objects have the same name. There may be more than one individual with the name "A. Smith."

To prevent the user from obtaining erroneous data, the code dictionary will have an ambiguity marker next to "A. Smith" and a reference to an output message that asks the user whether he is interested in A. P. Smith or A. B. Smith. In this instance, data about A. P. Smith are desired, so the user will rephrase his command.

Both the ambiguity and synonymy of words must be detected manually by input personnel or users. Their information is used to modify the code dictionary.

Now, suppose the user decided to find out more about Acme Electronics, that he wished to have a list of all its employees who are system engineers. He would try first to find those data that are explicitly in the file. His request might be programmed as follows:

1.1. LET ALPHA = (X) SUCH THAT (X/"AFFILIATED WITH"/"ACME ELECTRONICS CORPORATION") AND (X/"HAS PROFESSIONAL SPECIALTY"/"SYSTEM ENGINEER")

1.2. PRINT ALPHA, "HAS PROFESSIONAL SPECIALTY SYSTEM ENGINEER"

Statement 1.1 defines a set, which has been given the arbitrary name Alpha, composed of all individuals x for whom the conjunction of sentences

x/AFFILIATED WITH/Acme Electronics Corporation

and

x/HAS PROFESSIONAL SPECIALTY/system engineer

appears in the file. The sentences shown in Sec. III would result in the inclusion of the element (A. P. Smith) in set Alpha.

Statement 1.2 commands printout of one sentence for each element in the set Alpha. The sentence corresponding to the element above would be:

A. P. SMITH HAS PROFESSIONAL SPECIALTY SYSTEM ENGINEER

The command in statement 1.1 specifies a basic logical and set-theoretic operation, sometimes called set-abstraction. In traditional notation it would be written as

Alpha = {x|(x AFFILIATED WITH Acme Electronics Corporation)
& (x HAS PROFESSIONAL SPECIALTY system engineer)}

The programming language also permits definition of a set by specification of its members. For example,

1.3. BETA = SET (("A. P. SMITH"), ("R. E. JONES"), ("A. B. BLACK"))

Returning now to our user, it may be that he would like to find out how many of Acme Electronics' system engineers have graduated from UCLA. To begin his search, the command

1.4. LET GAMMA = (Y) SUCH THAT (Y/"GRADUATED FROM"/"UCLA")

would form a set comprising all individuals, y, for whom the explicit datum

y/GRADUATED FROM/UCLA

appears in the file. But note that the sentences about A. P. Smith given in Sec. III do not explicitly include one using the relation "GRADUATED FROM," while there do appear the pair of sentences "S1" and "S2" from which such a sentence follows by a trivial inference. The user can obtain that implicit data through a pair of additional commands:

1.5. LET DELTA = (Z) SUCH THAT (FOR SOME W)
(Z/"RECEIVED DEGREE"/(W/"AWARDED BY"/"UCLA"))

1.6. ZETA = JOIN (GAMMA, DELTA)

Statement 1.5 implements the user's inference that if a person received some degree "w" from UCLA, then he graduated from UCLA. The "triviality" of this inference to a human being should not obscure the fact that it must be stated explicitly to the machine, and it should illustrate why inference execution must be an essential part of any flexible data retrieval system, even one not primarily intended for logical analysis.

Statement 1.5 also illustrates the use of quantifiers--"for all" and "for some." In this instance, the statement "FOR SOME W" means that the user is not interested in which degree the individual, z, received, as long as there was at least one degree.

Statement 1.6 directs that the set union (or "JOIN") of the two sets Gamma and Delta be formed to obtain the new set Zeta, which comprises the names of those individuals who are known to have graduated from UCLA.

The usual set operations--union, intersection, difference, and cartesian product--are each implemented in the programming system. There are also operations that order the sets--alphabetically, numerically, or chronologically; that count the number of elements in a set; and that find the first, i-th, last, largest, or smallest element.

The inference underlying statement 1.5 follows strictly from the definition of the relations "GRADUATED FROM" and the pair "RECEIVED DEGREE" and "AWARDED BY." It would certainly be desirable to save the user the trouble of specifying that inference each time he is interested in the relation "GRADUATED FROM." This can be done by including a special file about the relations among the relations in the data store.

It would indicate how a given relation could be derived from other relations. It might contain, for example, the fact that:

x/GRADUATED FROM/y ≡ (For some w)(x/RECEIVED DEGREE/(w AWARDED BY/y))

Then when the user employed the relation "GRADUATED FROM," the above fact would automatically be employed--without the user's explicit command--to expand the search request to include the statement on the right.

The file of data about the relations is called the intensional file, as distinguished from the extensional file, which contains the basic data sentences. The intensional file also permits a saving of space in the extensional file. The situation is analogous to storing subroutines to compute the trigonometric functions instead of storing the corresponding tables, for the intensional file permits many relational sentences to be derived when needed, instead of having to be explicitly stored.

The user's interest in the number of UCLA alumni among Acme Electronics' system engineers may be satisfied by adding the following commands to his program:

```
1.7.  THETA = MEET (ALPHA, ZETA)
1.8.  N = SIZE (THETA)
1.9.  PRINT "THERE ARE", N, "UCLA ALUMNI AMONG ACME
      ELECTRONICS' SYSTEM ENGINEERS"
```

Statement 1.7 defines a new set Theta that contains only elements that are in both Alpha and Zeta. (Theta is the set intersection of Alpha and Zeta.) Consequently, any individual in Theta must be both an Acme Electronics system engineer and a UCLA alumnus. Statement 1.8 sets

the variable N equal to the number of elements in Theta. Statement 1.9 provides the appropriate output.

Plausible inference has not yet appeared in our examples. Suppose, however, that the user were uncertain about whether all system engineers at Acme Electronics had been explicitly identified as such in the file. He might try to infer an individual's professional specialty from other evidence. There are many ways to do so, of course, but let us assume that the user decides to use the subject indexing of an individual's publications. He might proceed as follows:

```
2.0.  LET IOTA = (U) SUCH THAT (FOR SOME P) (FOR SOME Q)
          (U/"AUTHOR OF"/P) AND (P/"SUBJECT INDEXED AS"/Q) AND
          (Q/"SUBFIELD OF"/"SYSTEM ENGINEERING")
2.1.  PUT IOTA/"HAS PROFESSIONAL SPECIALTY"/"SYSTEM ENGINEER"
          IN FILE TEMP
2.2.  LET ALPHA = (X) IN FILE MAIN, TEMP
          SUCH THAT (X/"AFFILIATED WITH"/"ACME ELECTRONICS CORPORATION")
          AND (X/"HAS PROFESSIONAL SPECIALTY"/"SYSTEM ENGINEER")
```

In statement 2.0 the set Iota has been defined to contain all individuals who have written papers, p, that were indexed as relevant to a subject, q, that is a subspecialty within system engineering. Presumably, the user has subjectively assigned a personally acceptable degree of plausibility to the assertion that such individuals are system engineers. The assertion is certainly not strictly true.

Statement 2.1 enters a sentence

u/HAS PROFESSIONAL SPECIALTY/system engineer

in a working file--FILE TEMP--for each element, u, in the set Iota. Users are prohibited from adding to or altering the data in the main

file, thus they are provided with a separate working file, which may be searched as an extension of the main file, but whose contents are destroyed after each session. In addition to containing data generated in the course of a computation, it may contain hypothetical or auxiliary data added for particular studies, or the results of previous sessions.

Finally, in statement 2.2 the user has redone the operation of statement 1.1, but this time is searching both the main file and the working file, which contains the inferred sentences concerning system engineers.

V. STATUS AND PROSPECTS

The Relational Data File is intended to be a computer assistant in the logical analysis of large collections of data. It consists of a data file, whose initial capacity is 232,000 sentences in a relational information language, and a programming language, in which logical analyses may be specified for computer execution.

Thus far, more than 7000 input data forms have been completed, a program for translating from them to relational sentences has been written and checked out, code dictionary and file manipulation routines are operational, and the first file is about to be filled. The output programming language is being coded. An initial version of the system is expected to be operational during the fall of 1966.

A wide variety of problems--logical and linguistic, hardware and software, practical and theoretical--arise during implementation of such a system. Among the questions that must be studied carefully if computer systems for logical analysis of data are to become effective and practical are the following:

- The design of relational information languages.
  Where are they applicable? What features should they possess? What rules guiding their design can be established?

- The organization of extensional storage.
  What structure of multi-level storage devices will provide the great volume of storage and the high-speed access that are essential for effective logical analysis?

- The design of logical analysis programming languages.
  What should the basic operations be? What strategy of file search should be employed to minimize access times? How can the language be made convenient for the non-specialist user?

- o The organization of intensional storage.
  How can the properties of relations and stored inference schemes be best employed? What tactics should be used to expand a user's inference specification?

The initial implementation of the Relational Data File will operate off-line in a batch processing mode. However, the process of logical analysis is ideally suited for on-line man-machine execution. As experience is gained, we shall move toward such a mode of operation.

The promise of computer-assisted logical analyses that Vannevar Bush recognized 21 years ago may soon be realized. While the practical problems are considerable, none seems insuperable.

REFERENCES

1. Bush, V., "As We May Think," Atlantic Monthly, Vol. 176, pp. 101-108, July 1945.

2. Kochen, M., "Adaptive Mechanisms in Digital 'Concept' Processing," Discrete Adaptive Processes--Symposium and Panel Discussion, New York: AIEE, 1962, pp. 50-58.

3. Kochen, M., Some Problems in Information Science with Emphasis on Adaptation to Use Through Man-machine Interaction (2 vols.), IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., April 2, 1964.

4. Black, F., "A Deductive Question Answering System," Ph.D. thesis, Division of Engineering and Applied Physics, Harvard University, June 1964.

5. Cooper, W. S., "Fact Retrieval and Deductive Question-answering Information Retrieval Systems," Journal of the Association for Computing Machinery, Vol. 11, No. 2, pp. 117-137, April 1964.

6. Elliott, R. W., "A Model for a Fact Retrieval System," Ph.D. thesis, The University of Texas, May 1965.

7. Kugel, P., "A Data Structure for Data Retrieval," paper read at the ACM National Conference, Syracuse, New York, September 1962.

8. Kugel, P., "Contemplative Computers," paper read at AIEE Winter General Meeting, 1963.

9. Lindsay, R. K., "Inferential Memory as the Basis of Machines Which Understand Natural Language," Computers and Thought, E. A. Feigenbaum and J. Feldman (eds.), New York: McGraw-Hill, 1963, pp. 217-233.

10. McCarthy, J., "Programs with Common Sense," Mechanisation of Thought Processes, Vol. 1, London: Her Majesty's Stationery Office, 1959, pp. 75-91.

11. Raphael, B., "SIR: A Computer Program for Semantic Information Retrieval," Ph.D. thesis, Massachusetts Institute of Technology, June 1964.

12. Slagle, J. R., "Experiments with a Deductive Question-answering Program," Communications of the Association for Computing Machinery, Vol. 8, No. 12, pp. 792-798, December 1965.

13. Thompson, F. B., The Application and Implementation of Deacon-type Systems, TEMPO, General Electric Company, Santa Barbara, Calif., RM 64TMP-11, October 1964.

14. Travis, L. E., "Analytic Information Retrieval," Natural Language and the Computer, P. Garvin (ed.), New York: McGraw-Hill, 1963, pp. 310-353.

15. Simmons, R. F., "Answering English Questions by Computer: A Survey," Communications of the Association for Computing Machinery, Vol. 8, No. 1, pp. 53-70, January 1965.

16. Levien, R. E. and M. E. Maron, Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval, The RAND Corporation, RM-4793-PR, December 1965.

17. Uspenskii, V. A., "The Problem of Constructing a Machine Language for an Information Machine," Problems of Cybernetics II, A. A. Lyapunov (ed.), New York: Pergamon Press, 1961, pp. 356-371.

DOCUMENT CONTROL DATA

1 ORIGINATING ACTIVITY	2a. REPORT SECURITY CLASSIFICATION
THE RAND CORPORATION	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

A COMPUTER SYSTEM FOR INFERENCE EXECUTION AND DATA RETRIEVAL

4. AUTHOR(S) (Last name, first name, initial)

Levien, R. E. and M. E. Maron

5. REPORT DATE	6a. TOTAL No. OF PAGES	6b. No. OF REFS.
August 1966	35	17

7. CONTRACT OR GRANT No.	8. ORIGINATOR'S REPORT No.
AF 49(638)-1700	RM-5085-PR

9a AVAILABILITY/LIMITATION NOTICES	9b. SPONSORING AGENCY
DDC-1	United States Air Force Project RAND

10. ABSTRACT

Describes the Relational Data File, a computer-based data retrieval system now partly operational, capable of storing millions of facts and of retrieving data both directly and through logical inference. A large collection of information about cybernetics research was used as the data base. Facts are stored in the form of sentences in an artificial information language, each representing a binary relationship between two entities. Data are entered by means of special input forms; a translation program in a special programming language, FOREMAN, derives the relational sentences that are then internally coded and stored in the data file. An intensional file that stores facts about relationships permits many data sentences to be derived when needed, rather than stored explicitly. Counting space for all auxiliaries, one IBM 1301 disk holds 232,000 sentences in the information language. The system operates on an IBM 7044 computer.

11. KEY WORDS

Data processing
Information storage and retrieval
Command Control
Cybernetics
Language
Computer programs
Research and Development
Management
Development