pone the implementation of these elements in order to get a COBOL system "on the air." The elements of required COBOL thus define the minimum language acceptable in order to consider a system as a COBOL compiler.

Some users tend to want every elective feature changed in status, to become a required feature. Others realize that the inclusion of more features in a compiler automatically make the compiler larger, which has the tendency to make it compile more slowly. It is, of course, much easier to implement the most elaborate forms of COBOL on machines of large capacity and high speed. It is probably not reasonable to implement some of the elective features on the smallest computers for which COBOL will be provided. It is anticipated, therefore, that certain electives will change status, but not necessarily all of them, and that the changes of status will come slowly. Clearly any statement of this form represents the author's opinion which may not be shared by others.

Honeywell COBOL

Honeywell is hard at work implementing COBOL for the 400. The complement of equipment which it specifies is one Card Reader, four Magnetic Tapes, 2,048 words of Core Memory, and one Printer; the Compiler will take advantage of additional tape and memory. Initially the compiler will implement all required and many elective features of COBOL, and will eventually implement the extensions discussed in this paper. Honeywell is about to begin the implementation of COBOL for the 800 and 1,800. Since it is doing COBOL for three machines, it has chosen an implementation technique rather different from that used for FACT—a technique with which we expect to shorten delivery time and reduce cost to us for the compiler development. We prefer to wait until the compilers are working before we discuss these techniques.

Information algebra

By R. F. Clippinger

COBOL was developed by the COBOL Committee, which is a subcommittee of a larger committee called the "Conference on Data System Languages" which has an Executive Committee and, in addition to the COBOL Committee, a Development Committee. Within the Development Committee are two working groups called the Language Structure Group (whose members are Roy Goldfinger of IBM, Robert Bosak of SDC, Carey Dobbs of Sperry Rand, Renee Jasper of Navy Management Office, William Keating of National Cash Register, George Kendrick of General Electric, and Jack Porter of Mitre Corporation), and the Systems Group. Both of these groups are interested in going beyond COBOL to further simplify the work of problem preparation. The Systems Group has concentrated on an approach which represents a generalization of TABSOL in GE's GECOM. The driving force behind this effort is Burt Grad.

The Language Structure Group has taken a different approach. It notes that FACT, COBOL, and other current data-processing compilers are "procedure-oriented," that is, they require a solution to the data-processing problem to be worked out in detail, and they are geared to the magnetic-tape computer and assume that information is available in files on magnetic tape, sorted according to some keys. The Language Structure Group felt that an information algebra could be defined which would permit the definition of a data-processing problem without postulating a solution. It felt that if this point were reached, then certain classes of problems could be handled by compilers that would, in effect, invent the necessary procedures for the problem solution. One sees a trend in this direction in FLOW-MATIC, COBOL, Commercial Translator, and FACT, if one notes that a sort can be specified in terms of the input file and the output file, with no discussion of the technique required to create strings of ordered items and merge these strings. Similarly, a report writer specifies where the information is to come from and how it is to be arranged on the printed page, and the compiler generates a program which accomplishes the purpose without the programmer having to supply the details. The FACT update verb specifies a couple of input files and criteria for matching, and FACT generates a program which does the housekeeping of reading records from both files, matching them and going to appropriate routines depending on where there is a match, an extra item, a missing item, etc. A careful study of existing compilers will reveal many areas where the compiler supplies a substantial program as a result of having been given some information about data and interrelationships between the data. The Language Structure Group has, by no means, provided a technique of solving problems once they are stated in abstract form. Indeed, it is clear that this problem is not soluble in this much generality. All the Language Structure Group claims is to provide an information algebra which may serve as a stimulus to the development of compilers

with somewhat larger abilities to invent solutions in restricted cases. The work of the Language Structure Group will be described in detail in a paper which will appear shortly in the *Communications of the ACM*.

Basic concepts

The algebra is built on three undefined concepts: ENTITY, PROPERTY and VALUE. These concepts are related by the following two rules which are used as the basis of two specific postulates:

(*a*) Each PROPERTY has a set of VALUES associated with it.
(*b*) There is one and only one VALUE associated with each PROPERTY of each ENTITY.

The data of data processing are collections of VALUES of certain selected PROPERTIES of certain selected ENTITIES. For example, in a payroll application, one class of ENTITIES are the employees. Some of the PROPERTIES which may be selected for inclusion are EMPLOYEE NUMBER, NAME, SEX, PAY RATE, and a payroll file which contain a set of VALUES of these PROPERTIES for each ENTITY.

Property spaces

Following the practice of modern algebra, the Information Algebra deals with sets of points in a space. The set of VALUES is assigned to a PROPERTY that is called the PROPERTY VALUE SET. Each PROPERTY VALUE SET contains at least two VALUES ɯ (undefined, not relevant) and ɒ (missing, relevant, but not known). For example, the draft status of a female employee would be undefined and not relevant, whereas if its value were missing for a male employee it would be relevant but not known. Several definitions are introduced:

(1) A co-ordinate set (Q) is a finite set of distinct properties.
(2) The null co-ordinate set contains no properties.
(3) Two co-ordinate sets are equivalent if they contain exactly the same properties.
(4) The property space (P) of a co-ordinate set (Q) is the cartesian product $P = V_1 \times V_2 \times V_3 \times \ldots \times V_n$ where V_i is the property value set assigned to the ith property of Q.
Each point (p) of the property space will be represented by an n-tuple $p = (a_1, a_2, a_3, \ldots, a_n)$, where a_i is some value from V_i. The ordering of properties in the set is for convenience only. If $n = 1$, then $P = V_1$. If $n = 0$, then P is the Null Space.
(5) The datum point (d) of an entity (e) in a property space (P) is a point of P such that if $d = (a_1, a_2, a_3, \ldots, a_n)$, then a_i is the value assigned to e from the ith property value set of P for $i = 1, 2, 3, \ldots, n$. (d) is the representation of (e) in (P). Thus, only a small subset of the points in a property space are datum points. Every entity has exactly one datum point in a given property space.

Lines and functions of lines

A line (L) is an ordered set of points chosen from P. The span (n) of the line is the number of points comprising the line. A line L of span n is written as: $L = (p_1, p_2, \ldots, p_n)$. The term *line* is introduced to provide a generic term for a set of points which are related. In a payroll application the datum points might be individual records for each person for five working days of the week. These five records plotted in the property space would represent a line of span five.

A function of lines (FOL) is a mapping that assigns one and only one value to each line in P. The set of distinct values assigned by an FOL is the value set of the FOL. It is convenient to write an FOL in the functional form $f(X)$, where f is the FOL and X is the line. In the example of the five points representing work records for five days work, a FOL for such a line might be the weekly gross pay of an individual, which would be the hours worked times payrate summed over the five days of the week. An Ordinal FOL (OFOL) is an FOL whose value set is contained within some defining set for which there exists a relational operator $\mathscr{R}$ which is irreflexive, asymetric, and transitive. Such an operator is "less than" on the set of real numbers.

Areas and functions of areas

An *area* is any subset of the property space P; thus the representation of a file in the property space is an area. A function of an area (FOA) is a mapping that assigns one and only one value to each area. The set of distinct values assigned by an FOA is defined to be the value set of the FOA. It is convenient to write an FOA in the functional form $f(X)$, where f is the FOA and X is the area.

Bundles and functions of bundles

In data-processing applications, related data from several sources must be grouped. Various types of functions can be applied to these data to define new information. For example, files are merged to create a new file. An area set $/A$ of order n is an ordered n-tuple of areas $(A_1, A_2, \ldots, A_n = /A)$. *Area set* is a generic term for a collection of areas in which the order is significant. It consists of one or more areas which are considered simultaneously; for example, a transaction file and master file from an area set of order 2. Definition: The Bundle $B = B(b, /A)$ of an area set $/A$ for a selection OFOL b is the set of all lines L such that if

(*a*) $/A = (A_1, A_2, \ldots, A_n)$ and
(*b*) $L = (p_1, p_2, \ldots, p_n)$, where p_i is a point of A_i for $i = 1, 2, \ldots, n$,

then $b(L) =$ True.

A *bundle* thus consists of a set of lines each of order n where n is the order of the area set. Each line in the bundle contains one, and only one, point from each area. The concept of bundles gives us a method of conceptually linking points from different areas so that they may be considered jointly. As an example, consider

two areas whose names are "Master File" and "Transaction File," each containing points with the property Part Number. The bundling function is the Part Number. The lines of the bundle are the pairs of points representing one Master File record and one Transaction File record with the same partner. A function of a bundle (FOB) is a mapping that assigns an area to a bundle such that

(*a*) there is a many-to-one correspondence between the lines in the bundle and the points in the area;
(*b*) the value of each property of each point in the area is defined by an FOL of the corresponding line of the bundle;
(*c*) the value set of such an FOL must be a subset of the corresponding property value set.

The function of a bundle assigns a point to each line of the bundle; thus a new Master File record must be assigned to an old master file record and a matching transaction file record.

An FOB may be expressed in three equivalent ways:

(*a*) $F \equiv (f_1, f_2, \ldots, f_k, \ldots, f_m)$
(*b*) $F \equiv (q_1' = f_1, \ldots, q_k' = f_k, \ldots, q_m' = f_m)$
(*c*)
$$F \equiv \begin{cases} q_1' = f_1 \\ q_2' = f_2 \\ \ldots \\ q_k' = f_k \\ \ldots \\ q_m' = f_m, \end{cases}$$

where F is the FOB, q_k' is the kth property for points in the area assigned by the FOB, m is the number of properties, and f_k is a function of the q_{ij} for $i = 1, 2 \ldots, n$ and $j = 1, 2 \ldots, m$, where q_{ij} is the jth property of the ith area of the area set: n is the span of the lines comprising the bundle. In form (*a*) the f corresponding to every q' must be specified. In forms (*b*) and (*c*) any expression $q_k' = f_k$ may be missing, in which case $q_k' = q_{nk}$ will be understood, where q_{nk} denotes the value of the property q_k for points in the nth area, i.e. the area which contains the last point of each line. If none of the f's is specified, i.e. $F \equiv (\ )$, then $q_k' = q_{nk}$ for $k = 1, 2, \ldots, m$. The area A assigned by an FOB F to the bundle B may be expressed as either:

$$A = F(B)$$
$$A = F(b, /A)$$
$$A = F(b, A_1, A_2, \ldots, A_n).$$

In summary then, bundles and functions of bundles permit the grouping together of one record from each of several files, and the computation of some new record as a function of the grouping records.

Glumps and functions of glumps

If A is an area and g is an FOL defined over lines of span 1 in A, then a *Glump* $G = G(g, A)$ of an area A for an FOL g is a partition of A by g such that an element of this partition consists of all points of A that have identical values f or g. The function g will be called the *Glumping Function* for G. The concept of a glump does not imply an ordering either of points within an element, or of elements within a glump. As an example, let a back order consist of points with the three non-null properties: Part Number, Quantity Ordered, and Date. Define a glumping function to be Part Number. Each element of this glump consists of all orders for the same part. Different glump elements may contain different numbers of points. A function of a glump (FOG) is a mapping that assigns an area to a glump such that

(*a*) there is a many-to-one correspondence between the elements of the glump and the points of the area;
(*b*) the value of each property of a point in the assigned area is defined by an FOA of the corresponding element in the glump;
(*c*) the value set of the FOA must be a subset of the corresponding property value set.

Consider again the back order file and include unit price with each part number. We can now define an area that is an FOG of the back order file in which each point has three non-null properties: Part Number, total back order quantity for each part, and total cost of this quantity. An FOG may be expressed in three equivalent ways

(*a*) $H \equiv (f_1, f_2, f_3, \ldots, f_k)$
(*b*) $H \equiv (q_1' = f_1, q_2' = f_2, \ldots, q_k' = f_k)$
(*c*)
$$H \equiv \begin{cases} q_1' = f_1 \\ q_2' = f_2 \\ \ldots \\ q_k' = f_k \end{cases}$$

where H is the FOG
f_i is an FOA
q_1' is the ith property for points in the area
k is the number of properties.

In form (*a*), the f corresponding to every q' must be specified. In forms (*b*) and (*c*), any expression $q_i' = f_i$ may be missing, in which case $q_1' = \mathfrak{u}$ will be understood.

Ordering of areas

Introducing area gave us a concept analagous to that of files. Bundling gave us functions similar to matching. Glumping gave us a way of handling operations such as summarizing within one file. In all three of these concepts, the notion of "order" was ignored. However, in some applications order is required. An Ordering $O(f, A)$ of an area A by an OFOL f of span 1 is a set of points $(p_1, p_2, \ldots, p_n)$ chosen from A such that the set exhausts A and either

$$f(p_i) < f(p_{i+1}) \text{ or } f(p_i) = f(p_{i+1}).$$

Operations of FOL's

For convenience, a series of operations are defined, namely concatenation, addition, multiplication, division,

negation, or/and, not, equals, less than, if—otherwise. For example, concatenation is defined as follows: Concatenation (o) is a binary operator with properties that:

(*a*) If $f_1 \circ f_2 = f_3$, then, for any L, $f_3(L)$ is the couple $f_1(L), f_2(L)$.
(*b*) If $f_1 \circ (f_2 \circ f_3) = f_4$, then $f_4(L)$ is the triple $f_1(L), f_2(L), f_3(L)$.
(*c*) $f_1 \circ (f_2 \circ f_3) = (f_1 \circ f_2) \circ f_3 = f_1 \circ f_2 \circ f_3$.
(*d*) In general $f_1 \circ f_2 \neq f_2 \circ f_1$.

The concatenation function is used for adjoining values of distinct properties, such as considering several sort keys side by side as one larger sort key. The Language Structure Group has used the Information Algebra in its present form to state several simplified data-processing problems, such as a payroll problem, and has also used the Algebra to define the concept of updating a file by several other files. It has noted that the information contained in a file may be represented in a property space in different ways depending on the choice of entities, and has shown that in certain cases functions of glumps and functions of bundles may be used to represent one area set in terms of another, and vice versa. The Language Structure Group plans to search for classes of problems which can be stated in terms of the Information Algebra, for which it can visualize automatic programs which could invent solutions to the problem and generate machine codes to influence the solution.

Discussion

Session 4: 18 April 1962

The Chairman (Dr. M. V. Wilkes, *University Mathematical Laboratory, Cambridge*): I have been asked to report briefly on the International Symposium on Symbolic Languages for Data Processing which was held recently in Rome (26–31 March 1962). This Symposium was organized by the International Computation Centre, which has just come into official existence after existing for some years in provisional form. The treaty establishing the centre was negotiated some years ago, but it has only recently been ratified by the minimum of ten Governments necessary for it to come into force.

The general situation in programming languages reminds me very strongly of the situation in computers themselves just ten years ago. At that time a few digital computers were working and others were in course of construction. Many people were fascinated by the new ideas behind the logical design of computers. It sometimes seemed that there was more talk than progress. So it is now with programming languages. A number of pioneering systems are in operation, but work on really sophisticated systems is only beginning and it will be some years before they are fully developed.

The Symposium was much dominated by ALGOL—perhaps too much so—and there was plenty of that type of arguing that one has come to associate with gatherings of ALGOL enthusiasts. The result was a background of controversy which resulted in there being, I thought, surprisingly little free discussion of programming languages in general, what features they should have, and what they are going to be like in the future.

It was clear that in one respect some advance in thinking had taken place since the ALGOL report was written, namely in regard to how a language should be specified. This morning Mr. Glennie referred to the fact that when FORTRAN existed for one machine only, the compiler constituted in effect the definition of the language, but that more formal treatment was necessary when the language came to be implemented for a number of machines. This brings out what I think has not always been clearly realized, that the defining of a language and the writing of a compiler are, in a way, similar operations. Thus, while the definition of a language should not be a compiler for any actual machine, there is much to be said for the view that a definition should be constructed along the same lines as a compiler. In ALGOL this was not done, and a sharp distinction was made between syntax and semantics. Many people now feel that there should not be two separate procedures, one for finding out whether a given piece of discourse is legal, and the other for finding out what it means, but that there should be one procedure only; this would either lead to the statement that the discourse means nothing, or would yield its meaning.

The discussions about ALGOL brought out two things quite clearly. In the first place ALGOL is incomplete in a number of respects. It lacks input and output facilities, and is not sufficiently rich in means for data description; one cannot, for example, declare double-length numbers, or manipulate sets. This need for extension is, I think, generally recognized. The other point relates to the efficiency of the object program, and here there is much controversy. Compared with other automatic-programming systems, object programs generated from ALGOL turn out to be rather long, and in view of certain features of the language, there is little that the writer of a compiler can do about this. A number of people, including myself, believe that some changes will have to be made in the language before it is acceptable for general use in a busy computing centre.

Unfortunately, when the ALGOL Committee finished their work in 1960, they did not set up any organization for performing the routine maintenance necessary with any programming language, or for developing and extending the language. For some time there has been in existence a movement to rectify this omission, and the matter had been considered in the week before the Symposium by the Council of the International Federation for Information Processing (IFIP), of which The British Computer Society is a member. The Council agreed that it would offer to set up a working party under IFIP auspices to take charge of the future of ALGOL. This offer was subject to the agreement of the original ALGOL Committee, and it so happened that quite a number of members of this Committee were present in Rome and able to meet together. I am glad to say that the IFIP proposal found acceptance, and the working party has now been set up. It includes the original ALGOL Committee, or as many of them as are still active in the subject.