24 April 1969

DATA DESCRIPTIVE LANGUAGES

by: Calvin N. Mooers

ABSTRACT

"Data" is defined as being digital matter having utility. A "data descriptive language" is a manner of description of how such digital matter can be transformed into a more useful form. Data descriptive languages are characterized as being iconographic, algorithmic, or a mixture of both. File structures are held to be merely a special case of data. An algorighmic data description is always possible, by definition, since otherwise the digital matter under consideration is not "data". In a trivial sense, such languages as COBOL or FORTRAN are found to be completely general data descriptive languages, although they are otherwise quite unsuitable because they are so poorly fitted to the descriptive task. A good data descriptive language should be congenial to use, general, transformable, and independent of supporting hardware or software. The value of small one-man projects for the creation of data languages is suggested.

* * *

The word "data" is misleading and pejorative. As a word, it tends unnecessarily to limit and to narrow the scope of consideration. A term which is not so limiting in scope is the term "digital matter". As a general conceptual starting point, digital matter can be taken as a linear sequence of bits (zeroes and ones). We note that this conception is not inherently limiting. Analog matter can be digitalized, picture matter can be "TV scanned" and digitalized, text messages can be coded, tree structures can be traced out into linear sequences, parallel codes can be serialized, and so on. In addition, the temporal aspects of digital sequences can also be treated by suitable conventions. Parallel "clock" pulses can be used to mark bit locations in asynchronous transmission, or the convention can be taken of filling quiescent periods with zeroes. With this viewpoint, a piece of digital matter has only one parameter, namely its length in total number of bits.

In order to have any utility whatsoever, a ditigal sequence must be unravelled or decoded. Conversely, in order to map any circumstance in the real world into such a digital sequence, some mode of coding and layout of the digital bits must be employed.

Broadly speaking, a "data descriptive language" provides the means by which it is possible to accomplish the two aspects just mentioned. That is, it 1) permits a description of how to analyze a linear digital sequence into a more useful form, and 2) permits a description of how to take content as presented by a real world situation and to map or synthesize it into some linear digital sequence.

There will be a variety of data descriptive languages, with some of them being especially applicable to certain classes of digital sequences. By the same token, it is likely that any of the languages will find it awkward to treat other classes of digital sequences.

A data descriptive language can be mainly "iconographic" or mainly "algorithmic". An iconographic data description is one which provides some manner of stylized picture of the arrangement and nature of the elements of the data. The COBOL 'picture' is a simple and vivid example. Iconographic descriptions can be much more involved, as for instance the data format descriptions of the Library of Congress Project MARC. These are iconographic because they show by diagram and verbal description 'what the data looks like'.

On the other hand, algorithmic descriptions literally tell how to put the data form together, or how to take it apart. They do this by specifying a series of operative steps. A parsing algorithm, with its implicit steps, is an example of an analytical algorighmic data description. A file generator computer program is an example of synthetic algorithmic data description.

An iconographic description, to be used, must somehow be converted into either an analytic or synthetic algorithm. That is to say, a step-by-step process must be generated to do the analytical or synthetic job desired with respect to some class of digital sequences. In the end, such a step-by-step process may be represented by the object code of a program for a digital computer. Alternatively, the steps of data parsing or data formation can be carried out by hand operations--if one is sufficiently accurate and patient.

Several philosophical or 'meta' comments on data and data structures are appropriate at this point of the discussion. The first comment is that any iconographic description, or any algorithmic description,is itself reducable to a sequence of digital bits. Therefore, such a description is also "digital matter" -- and constitutes "data". By the same token, the binary code for a digital computer program is "data". Consequently, "data" can specify operations upon "data". There is no inherent distinction between data and program, or data and description.

The second comment concerns 'file structures' and 'logical structures'. The position taken here is that these are merely

larger aggregations of data. A further source of confusion is that files have both a physical and a logical aspect. In its physical aspect, a file is an aggregate of physical bits arrayed on a physical medium. Such an aggregate can be mapped into some serial arrangement, where the serialization is suitable to the nature of the physical record medium. Thus any physical file is reducable to a linear sequence of digital matter. In its logical aspect, a file is also a linear sequence of digital matter. Here the mapping which produces the linearization is guided by the hypothetical logical structure, i.e., by the stated interrelationships between the parts of the data, rather than by the physical coordinates of the record medium device.

In the end, to cope with a file structure, whether we are concerned with the physical or logical aspect, we merely deal with additional levels of data description. Insofar as a data descriptive language is concerned, the difference between what is usually considered to be "data" and what is usually considered to be a "file structure of data" is merely a matter of quantity of description--and not a need for new kinds of description.

The third comment is really a conjecture. It is my conjecture that any iconographic description can lead to an algorithmic description but that the converse is not necessarily true. However, the proof that there may not be an iconographic version for a given algorithmic description will probably turn upon the restrictions on the kinds of complexities that are allowed in the iconographic description. In any event, it appears that algorithmic data descriptions are inherently more powerful and general. They are also likely to be more opaque to the human user. Iconographic descriptions are likely to be easier to understand.

There now come two very important questions. First, is an algorithmic data description always possible? Second, does there exist an algorithmic language which can describe all data formats? Suprisingly, the answer to both of these questions is 'yes'. An examination of the reasons for these affirmative answers is very revealing.

The first question derives its answer from a direct consideration of what is ordinarily meant by "data". By data,people usually mean some sequence of digital matter which can be decoded or unravelled to provide a useful output. If a digital sequence has no possible decoding, no unravelling, no meaning, or no utility, it is ordinarily not considered to be "data". I submit that this is what we really mean by "data". Similarly, viewed from the synthetic side, "data" is that which results from some kind of reversible transformation from a useful or meaningful input. Reversibility is a requirement, because if the transformation is not reversible, the content that is put into the digital sequence cannot be recovered. It is lost. It becomes 'garbage' or'noise' in ordinary parlance. It is no longer data.

The most general kind of transformation or coding and decoding that can be performed is the class of transformations which can be provided by a Turing machine operating through a finite number of steps. Therefore, in order for digital matter to constitute data, the sequence of digital matter must provide a "decidable problem" for some Turing machine. In short, if a Turing machine in a finite number of steps can analyze a digital sequence into some useful output, or can perform the converse synthetic step, then the digital sequence constitutes "data".

Therefore, the first conclusion: Given digital matter which is data, an algorithmic data description is always possible. (By definition, that is what we really mean by "data").

Unfortunately, Turing machine programs (and their programming languages, if any) have not achieved any great popularity in algorithmic processes. This leads us to inquire whether some other language might be used. In fact, almost any programming language can provide a completely general data descriptive language! It will now be demonstrated that FORTRAN and COBOL are both general purpose algorithmic data descriptive languages.

In proof, we first observe that COBOL and FORTRAN both contain the following capabilities: 1) arrays or tables are permitted whose elements or components can be designated by a numerical or symbolic index, 2) there is the ability to read an element of an array and to discriminate whether the element contains either a zero or one, 3) there is the ability to write either a zero or one into any element of the array, and 4) there is the ability to make decisions and to take actions in accordance with the state of the machine. We shall also presume that it is always possible to map any linear sequence of digital matter bit-by-bit into such an array. Clearly, we have here a Turing machine. It can operate upon the bits of digital matter, and it can perform any specified data transformation. Accordingly, we can conclude that either of the languages COBOL or FORTRAN is in fact a "general purpose data descriptive language." Moreover, both are 'standard languages' in that both are standardized by the U.S.A. Standards Institute.

Yet, something is wrong! Who wants FORTRAN or COBOL as his standard data descriptive language? They are just not right for the task. What do we mean by "right"?

It is my belief that what we intend, when we think of a 'good' data descriptive language, is a language which is more nicely suited than these languages are to the human requirements of describing useful kinds and arrangements of data. Our lack of enthusiasm for COBOL and FORTRAN is not suprising. COBOL and FORTRAN were devised especially for business and scientific numerical computations. The language desired should fit the new kind of problem. It should be more natural to use in the

data descriptive task. The language might be either algorithmic or iconographic. In my view, it could with benefit be a mixture of both. There probably should not be only one data descriptive language. Different kinds of languages will be found most suitable for different classes of data arrangement. These views can be summarized by saying that a good language should be "congenial" -- congenial for people to use.

The languages should also be "general", in the sense that no data form is precluded by the language. Some data forms, for some languages, will admittedly be awkward to handle. Generality is easily attained, as the previous discussion indicates, by merely giving a language the elementary features required to provide it with a universal Turing machine capability.

Data descriptive languages should be independent. Their use should not imply nor require the use of some particular brand of apparatus. Neither should it depend upon some particular brand of operation system or particular brand of popular language. Thus they should be "machine- and software independent". As an ideal, data descriptions, together with data sequences, should allow convenient and unlimited interchange and communication among the universe of potential users.

A language should be "transformable". By this is meant that a data description in one of the languages is transformable into equivalent descriptions, or usable forms, in other languages. Transformability may be the most difficult requirement to achieve.

One of the problems of transformability is illustrated by the difficulty of achieving a transformation from a set of assembly language statements to an equivalent set of source language statements in a language such as COBOL. For example, how does one locate and put back together all the many bits and pieces generated by a typical complex COBOL statement? What if program optimization in the compiler has cleverly eliminated certain redundant parts between several statements?

A collateral aspect of transformability is that the data descriptive languages should be usable for the automatic generation of programs (e.g., in ALGOL, COBOL, or FORTRAN) for the utilization of the data so described.

Data descriptive languages having the desirable characteristics named--languages which are congenial, general, transformable, and independent--do not yet exist. This uncomfortable fact poses the problem of how to achieve such languages. I am personally most pessimistic about any success from a typical American highly-funded crash program "to produce such a language by the end of the year". Neither do I see much hope from committee creativity. The problems are too delicate, the interplay of elements is too complex.

As a first technique to secure such languages,and as an interim measure, I would suggest seeking languages meeting a set of limited goals. A limited language, available early, could be immensely valuable.

As a second technique, both for gaining an immediate language, and for the solution of the more massive and general problem, I would recommend a plan of sustained effort from a half-dozen small independent projects. Each project would be essentially of a one-man character. From time to time the projects would meet to compare results. An annual review would keep them on their toes. I feel that this approach would have the greatest chance of success. A task of this sort cannot be hurried. Neither can it be done by immature students, though they may provide some new ideas. Support for the workers must be of a kind that provides some assurance of continuity for at least three or four years.

The cost of this approach--a group of small one-man study projects--is trifling in comparison with the visibly imminent disastrous costs of data format confusions and incompatiblities just ahead. It is my belief that the governmental agencies now embarking on the numerous vast and expensive investments in data banks and networks must find some way through joint effort to fund such an investment in data languages and techniques. If they cannot see the need, and find some way, then we can clearly doubt their planning competence and their custodianship of the public trust.

CONCLUSION

The discussion has shown that, in a trivial sense, we already have a number of algorithmic data descriptive languages of complete generality. However, they are not really adequate to the data descriptive task mainly because they are not congenial for people to use, and they lack the requisite transformability for them to be technically useful. The production of one or more adequate data descriptive languages is a creative challenge, and a professional necessity.

* * *