APPLIED DATA RESEARCH, INC.

LAKESIDE OFFICE PARK, WAKEFIELD, MASSACHUSETTS 01880 · 617 245 9540

AMBIT/G AS AN IMPLEMENTATION LANGUAGE

by

Carlos Christensen and
Michael S. Wolfberg

CA-7103-0211
March 2, 1971

This is the summary of a talk given in the session "Manufacturing Software, the Case for High Level Languages", J. W. Poduska, Chairman, at the IEEE International Convention and Exposition, New York, March 22-25, 1971.

AMBIT/G AS AN IMPLEMENTATION LANGUAGE*

Carlos Christensen and Michael S. Wolfberg
Corporate Research Center
Applied Data Research, Inc.
Wakefield, Massachusetts

The AMBIT/G programming system is, first of all, a high level system for the construction of software. The term "high level" is often applied to a programming language to indicate the use of some combination of English and mathematical notation. We intend a more general use of the term. In our broader sense, a successful high level system provides a complete framework of concepts and techniques for programming in addition to a language; that is, it channels and supports the thoughts of the programmer as well as his utterances.

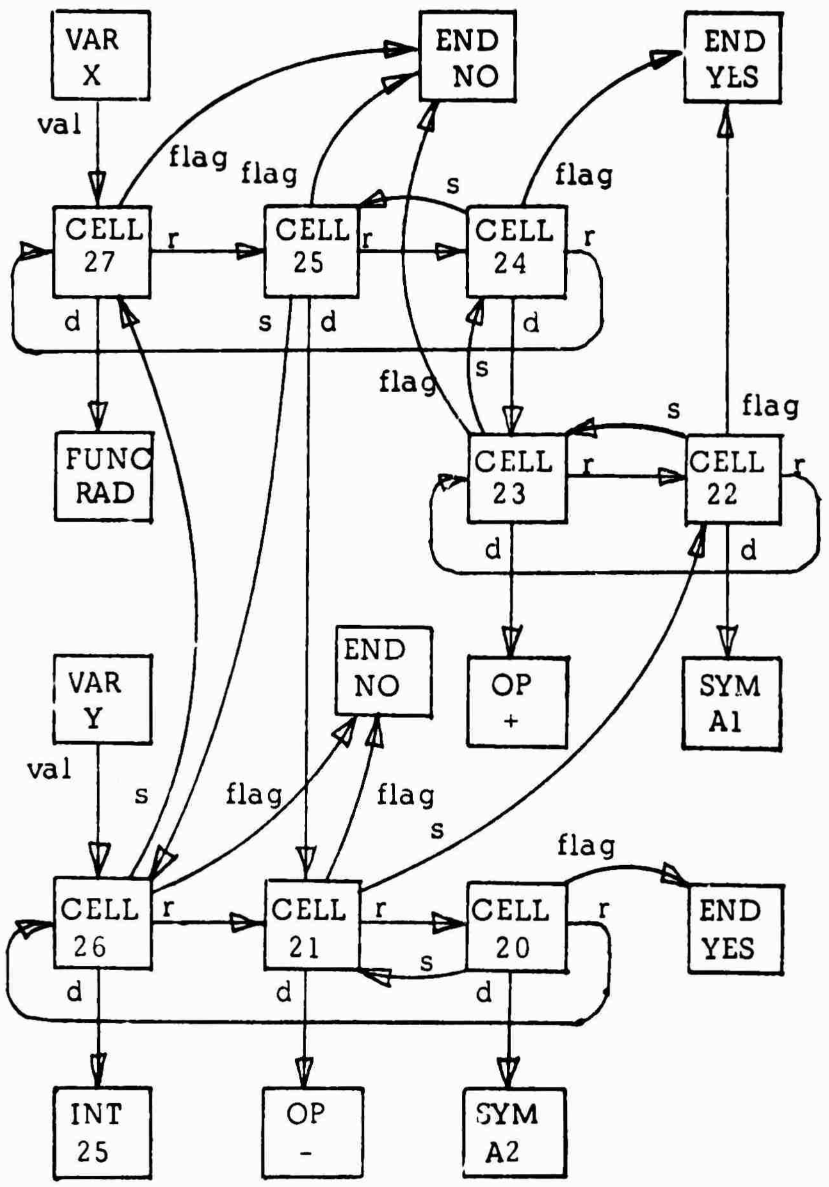
Our work on AMBIT/G has a simple underlying assumption. We believe that the characteristic activity of software construction is the design and use of complicated data structures, such as stacks, queues, rings, lists, and special tables. Indeed, the most important "construction" activity seems to be the structuring of data rather than programs. Accordingly, AMBIT/G is data-oriented to an unprecedented extent. At the beginning of a new programming task, the AMBIT/G user establishes a formal and 'machineable' statement of the representation and properties of his data. Only when his data design is complete does he begin programming.

English and algebra, as used in COBOL, FORTRAN, and PL/I for example, are an effective combination for commercial and scientific programming. However, these textual, essentially linear notations are not a natural medium for the description of structure in general or software data structures in particular. AMBIT/G rejects these notations in favor of another high level medium, the diagram.

*Research supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense under ARPA Order No. 1228.

The expository value of a diagram is well known. Flow charts of programs are very familiar and (more relevant to the present discourse) informal diagrams of data have been used for years to supplement program documentation. On the other hand, the formal adoption of a diagram as the "actual" data is quite unique to AMBIT/G and has a powerful effect; the diagram becomes an almost machine-like object, changing frequently in certain places and relatively fixed in others, a passive machine operated by a program but subject to its own built-in constraints.

An early use of informal data diagrams was in the representation of LISP lists, and many variations have since been used in papers on software. We obtained a formal model for data by restricting and simplifying the notation rather than elaborating it. The final result is a precisely defined form of diagram called a data graph. The following diagram is an example of a (small) data graph.



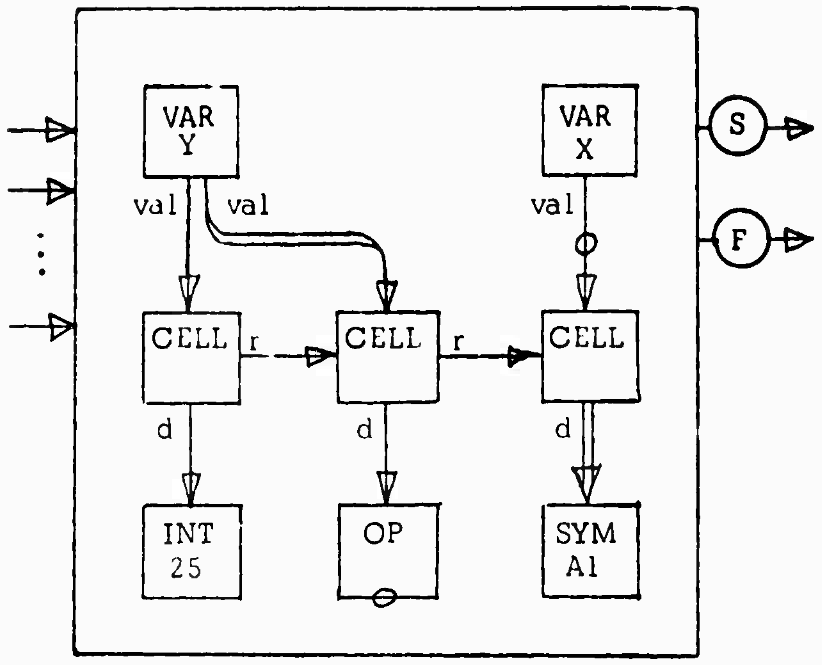
The diagram is composed of nodes and links. A node is a rectangle with a node-name written inside; this node name is a type written above a subname. There may be many nodes of a given type, and these are distinguished from one another by their subnames. In the diagram above for example, there are eight nodes of type CELL; their subnames are the integers from 20 through 27. A link is a line which begins at an origin node, passes close to its link name, and ends (with an arrowhead) at a destination node. Every node of a given type has a similar set of links. For example, every CELL in the diagram is the origin of four links which are named flag, r, s, and d, and every SYM is the origin of no links.

The types, subnames, and link names used in the data graph are selected by the programmer for each particular program. It is the facility for building special data structures, not the structures themselves, which is built into the AMBIT/G system.

Every data graph must be functional, that is, a given node name (as origin of a link) and a given identifier (as link name) must specify no more than one node name (as destination of the link). This allows the unambiguous specification of a "walk" along the links of a diagram by giving a starting node name and a sequence of link names. Purposeful link walking is an important activity of software programs. The data graph must also be permanent; that is, nodes and links cannot be created or destroyed during program execution. In fact, the only permitted operation is the "swinging" of a link so that its pointed end moves from one node to another

Once the fundamental data representation has been established, certain superficial but useful "abbreviations" are introduced. For example, the type is dropped from within a node boundary and is indicated by giving the node boundary itself a distinctive shape. Or link names are dropped by establishing for each different link a characteristic point of origin on the node boundary. Such convenience notations make the diagram much more readable.

An AMBIT/G program is a collection of rules connected by flow lines as in a flow chart. Each rule is itself a diagram and uses a notation which closely resembles that of the data graph. An example of a single rule is as follows:



This rule is executed when "control" enters along one of the incoming flow lines at the left; and its execution results in control exiting to another rule along the success or failure flow lines to the right. The inside of the rule can be interpreted in three paragraphs, as follows:

> First frame the data graph as follows: Select VAR/Y, follow the val link, and call its destination c1. Is c1 a CELL node? Select c1, follow the d link, and answer: is its destination INT/25 ? Select c1, follow the r link, and call its destination c2. Is c2 a CELL node? Select c2, follow the d link, and call its destination o1. Select c2, follow the r link, and call its destination c3. Is c3 a CELL node? (Should the answer to a frame question be "no", you have detected the consequences of a programming error; take the day off and get undefined.)
>
> Next test the data graph as follows: Is o1 an OP node? Select VAR/X, follow the val link, and answer: is its destination c3? (Should the answer to a test question be "no", take the fail exit from the rule.)
>
> Finally, (if you haven't gone away) modify the data graph thus: Select VAR/Y and set its val link to point to c2. Select c3 and set its d link to SYM/A1. (No questions are asked during modification. When you are done, take the success exit from the rule.)

Every (single-line) link in any rule must be a part of an anchored walk. An anchored walk begins with a node whose full name (type and sub-name) is given in the rule and repeatedly "steps" from one node in the rule to another, each time following a link from origin to destination. This restriction means that the pattern-match can be implemented very efficiently; in fact, none of the "searching" characteristic of general pattern-matching is ever required.

Over the past five years a series of experimental implementations of AMBIT/G have been completed: first on an SDS 940 [1], then on the Lincoln Laboratory TX-2 [3], and recently on Multics [2]. Nevertheless, AMBIT/G has not reached the point at which it can be used to implement software; important practical and theoretical work remains to be done. On the other hand, AMBIT/G has spun off a more restricted but very practical language, AMBIT/L [4], which thus far has been used with success for two large-scale software projects and which shows promise of wide use.

References

1. Christensen, Carlos. "An Example of the manipulation of directed graphs in the AMBIT/G programming language", In Interactive Systems for Experimental and Applied Mathematics, M.Klerer and J. Reinfelds, eds., Academic Press, New York, 1968.
2. Christensen, Carlos; Fischer, Michael; and Wolfberg, Michael S. "Final report for the project 'Research in Machine-Independent Software Programming'.", Massachusetts Computer Associates, Inc., Wakefield, Mass. February 1971.
3. Rovner, Paul D. and Henderson, D. Austin. "On the implementation of AMBIT/G: a graphical programming language", Proceedings of the AFIPS/ACM International Conference on Artificial Intelligence, Washington, D.C., May 1969.
4. Christensen, Carlos. "An introduction to AMBIT/L, a diagrammatic language for list processing", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March 1971.