Letters to the Editor

Thoughts on Continuous Change Simulation Languages

EDITOR:

Authors Teichroew and Lubin [*CACM 9*, 10 (Oct. 66)] deserve credit for their excellent paper on Simulation Languages. As a member of "the other camp," which is concerned with "conftinuous systems simulation languages," I am particularly grateful lor the insight gained from the analysis of discrete event simuators.

The authors included a brief discussion of "continuous-change simulation languages" and gave reference to the appropriate literature on the subject. I should like here to add some thoughts on this topic, within the framework of the subject paper. First, let me comment that the Simulation Software Committee of the Simulation Councils, Inc. (an AFIPS member) was formed in 1965 for the express purpose of preparing language standards for the class of simulation languages it has chosen to call "continuous system simulation language" (CSSL). As noted by Teichroew and Lubin, there have been many such programs developed since the first one in 1957—the count is at least 23. The committee expects to publish the completed standard this spring.

It is customary in casual discussion to distinguish between the two classes of languages by use of the terms "continuous" and "discrete" simulations. While it is true that these words characterize the typical models represented in the two kinds of languages, I conclude from these authors that such a distinction is not fundamental to the structure of the language, given appropriate programming or "activity subroutines." I suspect that CSL can approximate continuous simulation, and that a present-day CSSL certainly can represent discrete behavior.

The distinction that *is* fundamental is characterized by these excerpts:

> CSSL: the system simulation consists of "a continuous flow of information or *material counted in the aggregate rather than individual items.*"
>
> "Discrete" Simulators: "*items* flow through the system." "This type of simulation consists. . . in *keeping track of where individual items are,*" (italics mine)

It is possible with CSSL to represent flow of discrete items through a system, as well as queueing and actions that are conditional upon the size of the queue. However, the flow of items must be homogeneous: individual items cannot be distinguished; core space is not required for all items of a queue, only the current size of the queue is retained.

The authors have taken care in clarifying the terminology of the languages analyzed. Moreover, they have suggested a basic set of terms, in the legends of the tables. Looking at these from a different point of view, I detect a conflict in the definition of what is being simulated, i.e., the "simuland," to use the term proposed by John McLeod, editor of *Simulation* magazine. The authors say: "There appears to be general agreement that the distinction between objects being simulated, properties of these objects, data describing the environment, and lists of objects having a particular property is useful in formulating a simulation model and should be retained in future discrete-change simulation languages. However, the mechanics of this definition can become burdensome." In Table III.1, item 1 is called the "object being simulated: fundamental element (Record)." These are the discrete entities that flow through the model (which are not representable as distinct entities in a CSSL). Next, item 4 is "data about the environment (variables)." Elsewhere these are referred to as "data about the *simulated world.*" I submit that it is clearer (and perhaps more accurate) to speak of the simuland as the simulated system (the model and its environment; or the process) which has variables and parameters; the latter being changed from case to case. The objects, or entities, that enter the simulated system are simulated input data that are processed by the model. These are analogous to "forcing functions" in CSSL terminology.

T. D. TRUITT
Electronics Associates, Inc.
Princeton, New Jersey 08546

Böhm and Jacopini's Reduction of Flow Charts

EDITOR:

In the first part of the paper by Böhm and Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules" [*Comm. ACM 9*, 5 (May 1966)], it is proved that any program may be mechanically transformed into an equivalent program whose flowchart is "decomposable into $\Pi\ \Phi\ \Delta$ ". This last phrase means that all loops are properly nested; this concept is equivalent to the block form I have defined in "Some Transformations and Standard Forms of Graphs, with Applications to Computer Programs," to be published shortly (in *Machine Intelligence 2*, D. Michie, ED., Oliver and Boyd, Edinburgh). However, even by making the same assumptions as Böhm and Jacopini a stronger reduction than theirs is possible.

In order to prove their result, Böhm and Jacopini introduce new Boolean variables into the program (or equivalently a single Boolean stack, but a bound for the maximum depth required for this stack may easily be given). As they point out, it is not usually necessary to add a new concept such as "Boolean variable" because in most applications we would expect to find some existing concept which would serve the same purpose. However, for simpler exposition we assume new variables have been added. New predicates to test these variables are also needed, together with assignments to set them true or false.

If we allow these additions to a program, then it is clear that any two nodes (P and Q) of the directed graph which is the program's flowchart may be coalesced into one new node N. The node N has as input all the arcs leading into P and also all those leading into Q, and similarly for the outputs. Loops PP or QQ do not effect this argument. A new Boolean variable, B_N , is introduced and instructions added to set it true on all arcs originally leading into P, and false on those originally leading into Q. By testing this variable the correct output arc of N may be chosen. By repeating this process, all the nodes of a program may be collapsed into a single node; the resulting program will have a trivial flowchart consisting of node A with loops AA and arcs to A from the input and from A to the output. In Böhm and Jacopini's terminology we have reduced the program to one whose flowchart is decomposable into Π, Φ, and Δ, with at most one Φ.

The result may be illustrated using ALGOL as follows: Let L_0 and L_n be labels on the start and exit, respectively, of a program P, and let L_1 , $\cdots$, L_{n-1} be all the other labels. For $0 \leq i \leq n-1$ and $0 \leq j \leq n$, let $P_{i,j}$ be the condition for control to pass from L_i to L_j

(*Letters are continued on page 473*)

practical limit since higher-dimensional objects are presently too detailed to be displayed adequately by the computer.

Discussion

At first it was thought that the computer-generated movies of the four-dimensional hyperobjects might result in some "feeling" or insight for the visualization of a fourth spatial dimension. In particular, perhaps some visualization of a solid four-dimensional hyperobject would be gained from the distortions in the three-dimensional perspective projection. Unfortunately, this did not happen, and we are still as puzzled as the inhabitants of Flatland in attempting to visualize a higher spatial dimension.

However, the importance of the techniques presented in this paper is the use of a digital computer to generate visual displays of the three-dimensional projections of the hyperobjects. Such displays of rotating hyperobjects could be produced most efficiently by a computer since the projections and drawing would be too tedious and impractical to produce by any other method. Although no actual mental visualization of the fourth dimension resulted from the computer-generated displays, it was at least possible to visually display the projections and be puzzled in attempting to imagine the rigid four-dimensional hyperobject. Of course, these techniques should be useful in displaying data with more than three variables.

The movies have already been useful in extending knowledge of three-dimensional perspective projections to higher dimensions. The techniques have been applied to real-time graphical displays so that the user can rotate, translate, and manipulate hyperobjects and hyperdata and immediately see the results on a graphical display.

Acknowledgments. Grateful acknowledgment is made to Dr. D. E. Eastwood, Dr. M. V. Mathews, Dr. M. R. Schroeder, and Dr. M. M. Sondhi for their lively discussions, enthusiasm, and mathematical assistance in the multidimensioned aspects of this hyperdimensional project.

RECEIVED FEBRUARY 1967; REVISED APRIL, 1967

REFERENCES

1. ABBOTT, EDWIN A., *Flatland.* Dover Publications, Inc., New York, 1952.
2. BOERNER, HERMANN. *Representations of Groups.* North-Holland Publishing Co., Amsterdam, 1963.
3. COXETER, H. S. M. *Regular Polytopes.* The Macmillan Co., New York, 1963.
4. KENDALL, M. G. *A Course In the Geometry of n Dimensions.* Hafner Publishing Co., New York, 1961.
5. MURNAGHAN, FRANCIS D. *The Unitary and Rotation Groups.* Spartan Books, Washington, D. C., 1962.
6. NOLL, A. MICHAEL. Computer-generated three-dimensional movies. *Comput. Autom. 14,* 11 (Nov. 1965), 20–23.
7. SOMMERVILLE, D. M. Y. *An Introduction to the Geometry of N Dimensions.* Dover Publications, Inc., New York, 1958.
8. STROMBERG, GUSTAF. Space, time, and eternity. *J. Franklin Inst. 272,* 2 (Aug. 1961), 134–144.

LETTERS—Continued from p. 463

without passing any other label, and let $S_{i,j}$ be the sequence of assignment statements obeyed on this path. Define new Boolean variables $B_0, B_1, \cdots, B_n$. Then P is equivalent to the program:

```
START: B_0 ← true;  B_1 ← false;  ··· ;  B_n → false;
    L: if B_n then go to EXIT;
       if B_0 ∧ P_0,0 then begin B_0 := false;  S_0,0 ;
         B_0 := true end else
       ...
       ...
       if B_i ∧ P_i,j then begin B_i := false;  S_i,j ;
         B_j := true end else
       ...
       ...
       if B_n-1 ∧ P_n-1,n then begin B_n-1 := false;  S_n-1,n ;
         B_n := true end;
       go to L;
EXIT:
```

This program has a trivial flowchart of the form indicated above.

Böhm and Jacopini are interested in reducing as far as possible the number of concepts used and it is then reasonable to code up previous flow of control into variables, as this can usually be done within the existing framework. If, however, one's motivation is to simplify the program's structure so that we may better answer questions such as whether the program loops indefinitely, then this coding of the control into variables is no help at all. It remains true though that the block form is a very natural standard form to use, and it is certainly possible to transform many programs into equivalent programs in block form without resorting to the coding of control features as values of variables. Some preliminary conjectures along this line are reported in my paper referred to above.

DAVID C. COOPER
Carnegie Institute of Technology
Pittsburgh, Pennsylvania 15213

A Comment on Galler's Letter

EDITOR:

I find Mr. Galler's letter to the membership [*Comm. ACM 10,* 5 (May 1967)] a well-intended guide to penetrating the ACM power structure. "Fred Jones," a typical programmer with some ideas about file structures, starts in Mr. Galler's account as an unknown, and rises until "he may even find himself a subcommittee chairman." This up-note ending is as unquestioned as that in a classic Hollywood movie, until Mr. Galler adds, "It could happen to almost anyone—it did to some of us." To me, that is an unwittingly frank statement of a menace.

Committees don't often discover anything. If Fred Jones' ideas about file structures are genuinely good, he should indeed spread them around, and he should listen to and benefit from the related ideas of others; the network of committees and meetings is admirably suited for this. But he should also pursue his ideas further, which might best be done by *not* finding himself a subcommittee chairman. With the surplus of "joiner" activities in the computer field, a line should be drawn as to how many to take part in.

If such a line is not drawn, as in Mr. Galler's otherwise excellent editorial, the creative accomplishments possible outside the mechanisms of "the establishment" of committees, meetings, and block-diagram power structures will be killed. In fact, some totally, perversely independent types, I believe, might contribute at least as much to computer science as do the unquestioning joiners of committees. Self-directed, intensive thought and research, as well as some mellow, not-geared-to-the-minute reflectivity, is needed in this computer game.

JERRY A. RALYA
Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543