Edited by Bernard P. Zeigler • *Weizmann Institute of Science*

Series introduction

The computer simulation enterprise has undergone successive waves of transformation (one might even say revolution) in its short existence. From the analog modelling of aircraft control systems and the Monte Carlo simulation of neutron beam trajectories to the comprehensive ambitions of "world simulationists" stretches a tremendous span of quantitative and qualitative technological developments. Indeed, rapid expansion of the domain of application of simulation has been accompanied by steady progress in the general methodology of simulation.

Flowing somewhat separately from all this is the systems modelling stream, with all its tributaries, e.g., general systems theory, systems science, systems approach. The time has arrived for a dialogue between the systems scientist and the simulationist. The concepts, models, and methods of systems theory are awaiting concrete exploitation. Conversely, the sheer magnitude of simulation modelling projects currently being contemplated calls for an organised attack on the complexity that systems modellers have been heralding for some time.

Papers in this series aim to bring the reader of *Simulation* in contact with developments in systems modelling which are particularly relevant to the use of simulation. The concepts and the models of systems theory attain their power and generality from their abstract mathematical cloak. While we cannot do full justice to these ideas in this series, we do hope to convey the essentials through illustrative examples with as little specialised terminology as possible. The style and length of articles in the series will parallel those of the *Simulation Today* series, emphasizing survey and exposition rather than first presentation of original research.

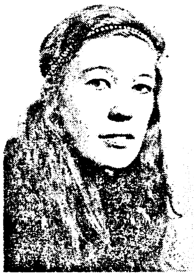
I wish to thank the editors of *Simulation* for opening up this forum, and I hereby turn to you, the readers, for submission of articles, comments, and suggestions.

The first article in the series, which begins on this page, deals with the interface between abstract formal models and high-level simulation languages. The formal models encourage a structured approach to simulation modelling. Conversely, the expressive power of high-level simulation languages reveals some of the limitations of formal models and stimulates the development of more appropriate concepts for structuring models. Besides, watching the computer generate lifelike forms is fun!

Bernard P. Zeigler, Editor

Simulating the growth of cellular forms

by
Pauline Hogeweg
Bioinformatica
Subfaculty of Biology
University of Utrecht
Padualaan 8
Utrecht, the Netherlands



PAULINE HOGEWEG received her university education at the University of Amsterdam (doctoral degree in 1969, major in biology). In 1976 she received her doctorate degree from the University of Utrecht (thesis: Topics in Biological Pattern Analysis*). Since 1970 she has been a staff member at the Subfaculty of Biology of the University of Utrecht, with her main field of research in bioinformatics. Her major interests include pattern generation and pattern recognition.*

INTRODUCTION

Organisms grow and develop their characteristic form by repeated cell divisions. It may be assumed that this process is regulated by the state of the individual cells and their interactions with cells adjacent to them, that is, in wall-to-wall or wall-fluid contact with them. The myriad patterns of intricate delicacy apparent in the development of plants and animals may seem to us marvelously complex; yet it is conceivable that such growth patterns may be generated by relatively simple rules which in the case of organisms we assume to be encoded in each cell's DNA. Current views attribute a small number of possible states to a cell and specify its transitions to other allowed states as a function of the state of the individual cell and the states of its *informationally adjacent* neighbours (i.e., those that affect it).[9]

The study of cellular growth models is an emerging field called *cellular auxology*,[9] and its principal investigative tool is observing the behaviour of the model by interactive computer simulation. Well-known examples of cellular growth models used in cellular auxology are based on cellular spaces (cellular arrays). By studying such systems we hope to gain insight into the process of cellular growth and pattern generation in simple biological systems.

Today we are still in the stage of formulating and investigating hypotheses that hinge on simple formal properties of cellular structures. We therefore ignore all properties other than those postulated, including many known to be relevant for the development of form in organisms (for example, mass-energy relations).

Sohnle, Tartar, and Sampson have reviewed cellular space models, their use in biology, and simulation systems generally available to study them.[8] This paper reviews some recent work on more general models of cellular growth. In all earlier work, cellular models were formulated in such a way that the transitions of all cells were forced to take place simultaneously. In this way global control of cellular events was implicitly introduced, thus restricting studies to a very limited class of models. In contrast we emphasise systems in which cells are granted various degrees of autonomy. Cell transformations are therefore not forced to be synchronous, but may be entirely asynchronous or locally synchronised to various degrees. These generalisations allow us to study a larger class of formal properties of cellular structures. These more general models can be conveniently formulated in advanced discrete-event simulation languages such as SIMULA/67.

CELLULAR SPACE SYSTEMS

The first computer experiments to study the growth of cellular forms were done by Ulam,[9] using the cellular space systems of von Neumann[10] in which the cells are *finite automata* (i.e., their next state is entirely determined by the current state and the total input, and there is a finite number of different states and transformation functions). Each cell receives inputs from a small number of other cells. There is a uniform rule to determine which cells constitute the *input neighbourhood*, i.e., do the sending, usually the cells immediately adjacent on a two-dimensional lattice. The next state of the cell is a function of its state and its inputs, the total input to any one cell being a function of the states of the neighbouring cells.

The spatial layout of the cells renders cellular space systems very convenient for experiments in pattern generation. Fascinating experiments in such systems include Conway's Game of Life[1] and the perplexingly simple "modulo prime" rule for self-replication (see Figure 1).[11]

When one simulates a cellular space system on a *sequential* computer, the following problem arises: As formulated *mathematically*, the cellular space is composed of an infinite number of cells (so as not to preclude unbounded growth). All cells operate in parallel so that an infinite amount of time is theoretically needed to complete the simulation of a single time step. The obvious way out of this dilemma is to use a finite lattice of cells. But a finite lattice does not allow simulation of patterns which expand beyond its original confines.

In 1976 Zeigler suggested a better way; it employs a next-event simulation strategy.[12] His algorithms take into consideration at each time step only those cells which can possibly change state; these are the cells which have changed state at the last time step and the cells they influence. All other cells cannot possibly change state, since neither they nor their neighbours have changed at the last time step. An arbitrary finite configuration of cells (limited only by the size and speed of the computer) may be processed in reasonable time, because only "active" cells (those changing state) are processed at each time step.

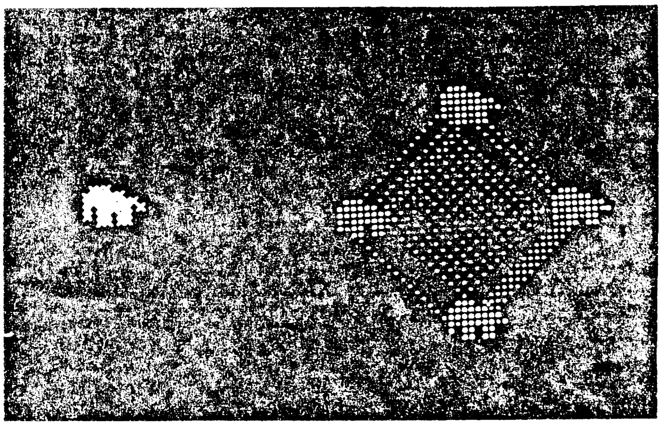


Figure 1 - Self-replication in cellular space systems. If the state of the cells in the next generation equals the sum of the states of its adjacent cells modulo the number of states (which is prime), any initial configuration will be replicated in a number of copies equal to the number of adjacent cells (Winograd)[11].

(a) Initial configuration

(b) 16th generation (4 elephants) and the previous generation in dots

A sketch of a restricted version of such a simulation algorithm is given in Table 1 using SIMULA/67 as the language for discrete-event simulation.* In the example, CELLS being processed are stored in an array CELLSTORE. Procedures HASHIN and HASHOUT do the storing and the retrieval. The arrays XN and YN hold the coordinates of the neighbours of CELL at the origin of the lattice. If the currently processed CELL has spatial coordinates X and Y, then the Ith neighbour has coordinates X+XN(I) and Y+YN(I) (i.e., the arrays XN and YN used in this way contain the uniform rule for obtaining the input nieghbourhood mentioned above) and the retrieved STATE is stored in INPUT(I). After applying the transformation rule (procedure TRANSFORM), CELLS in STATE Ø are dropped for further active processing and storage. Such deleted CELLS are regenerated by any neighbours entering a nonzero STATE. The simulation will be valid under the assumption that Ø is a quiescent state, i.e., if a CELL and all its neighbours are in STATE Ø then it will remain in STATE Ø. This strategy will be most efficient when the number of quiescent CELLS

*SIMULA is employed here because of its ability to capture in high-level statements the very features characteristic of discrete-event cellular space models. Detailed comments are provided for those unfamiliar with SIMULA/67; some familiarity with the underlying language ALGOL-60 is assumed, but most of these statements should be self-explanatory.

(CELLS in STATE ∅) is relatively large at each time step (as compared to the total of ever-active CELLS).

L-SYSTEMS

L-systems are cellular growth systems which were formulated to model pattern development in organisms.[2,3] There are two main differences between L-systems and cellular space systems. In L-systems *new cells are generated by division of old cells*, and *there is no uniform rule to determine the input neighbourhood of a cell* from its spatial coordinates as *the input neighbourhood is determined by a combination of position and ancestry*. Cellular space systems do not involve the creation of new cells but only changes in the states of the cellular spaces.

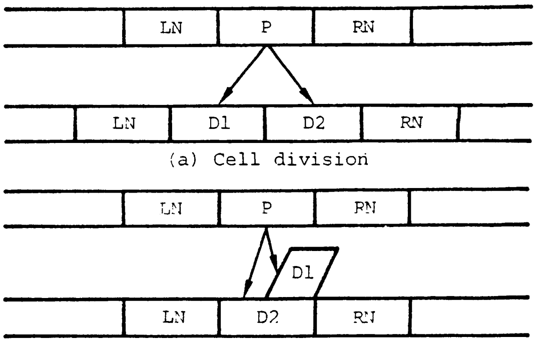
The simplest rule of the ancestry/position type applies to one-dimensional cell arrays. In this rule, if the parent cell divides into two cells, the daughter cells become neighbours of each other and each inherits one neighbour of the parent cell. Branching structures can be created by adding the possiblity of branching from the main stem; in that case one of the daughter cells inherits both neighbours of the parent cell, while the other has just the other daughter cell as neighbour and sticks out

```
REF (HEAD) ARRAY CELLSTORE[1:N];               /declares CELLSTORE as a pointer
                                                (called HEAD in SIMULA) to a list
                                                for storing the active CELL

REF (CELL) PROCEDURE HASHOUT(X,Y);             /procedure to retrieve a pointer to
                                                the CELL with coordinates X and Y
                                                from the CELLSTORE

PROCEDURE HASHIN(C); REF (CELL)' C;            /procedure to store the CELL with
       BEGIN...END;                             pointer C in CELLSTORE

INTEGER ARRAY XN,YN[1:NN];                     /arrays to hold the X and Y coordi-
                                                nate of the neighbors of the CELL
                                                at the origin (there are NN of
                                                them)

INTEGER PROCEDURE TRANSFORM(INPUT,STATE);      /procedure which contains trans-
       INTEGER ARRAY INPUT; INTEGER STATE;      formation rules
       BEGIN...END

PROCESS CLASS CELL(X,Y); INTEGER X,Y;          /the following code defines CELL as
                                                a SIMULA CLASS, and as subclass of
                                                the building CLASS PROCESS; an un-
                                                limited number of objects of a
                                                CLASS can be generated and can
                                                exist simultaneously in the sys-
                                                tem; each possesses the local
                                                variables of the CLASS definition
                                                (with its own values) and process-
                                                es its code when activated

       BEGIN INTEGER STATE,NEWSTATE;

       HASHIN (THIS CELL);                     /the parameter C of HASHIN points
                                                to the present CELL

NEXT:  FOR I:=1 STEP 1 UNTIL NN DO             /the Ith neighbor of the present
          INSPECT HASHOUT(X+XN[I],Y+YN[I]       CELL is INSPECTed, i.e., variables
          WHEN CELL DO INPUT[I]:-STATE          refer to the values of that CELL;
                                                in particular STATE is that of
                                                CELL neighbor

          OTHERWISE                            /if nc neighboring CELL is found

          BEGIN INPUT[I]:=∅

                IF STATE≠∅ THEN                /STATE refers here to the STATE of
                                                the present CELL

          ACTIVATE NEW CELL (X+XN[I],Y+YN[I])  /a new copy of CELL is generated and
                AFTER CURRENT                   will start execution just after
                                                this one is suspended
       END;

NEWSTATE:=TRANSFORM(INPUT,STATE);              /apply the transformation rule

HOLD(∅);                                       /HOLD is a procedure of the CLASS
                                                PROCESS which suspends the current
                                                process for a duration given by.
                                                its argument. To simulate simul-
                                                taneous processing, we specify a
                                                zero duration

STATE:-NEWSTATE; IF STATE≠∅ THEN BEGIN         /suspend processing for 1 time unit
HOLD(i); GOTO NEXT; END;

OUT;                                           /inactive CELLs are removed from
                                                CELLSTORE with the procedure OUT
                                                of PROCESS

       END;                                    /when passing the final END, the
                                                CELL is deleted from the system
```

Table 1-

Sketch of a discrete-event simulation of cellular space systems (coded in SIMULA/67 with extended comments) in the environment (see Figure 2). Table 2 shows how such branching structure-generating L-systems can be expressed in SIMULA/67.

Comparing the formulation of the cellular space system (Table 1) and the L-system (Table 2) we note that in both cases the system is characterised by the definition of the behaviour of one of its cells, given its preceding state and the then-current states of its informationalneighbours. The entire system is completely defined by an initial configuration of cells and the definition of each cell's behaviour. As major differences between the two definitions (for cellular space models and L-systems, respectively), we note the absence in the latter case of a globally accessible datastructure to store the CELLS (CELLSTORE of Table 1). Instead, the CELLS in L-systems possess pointers which identify their neighbours (LN and RN, for left neighbour and right neighbour in Table 2), and these pointers are established when a cell divides. (Cell death is not considered in the example, but could be incorporated in a straightforward way.) Upon cell division a new cell is generated (C:NEW CELL(LN,THIS CELL, 1), see Table 2).



(a) Cell division

(b) Cell branching

Figure 2 - Cell division and cell branching. The parent cell P divides into two daughter cells D1 and D2, inserted in the cellular structure as shown, by setting the pointer LN and RN of each of the cells involved to the appropriate cells. D1 is generated as a new datastructure (C); D2 is generated by modification of the datastructure of the parent cell.

```
INTEGER PROCEDURE TRANSFORM(LS,S,RS);
                 INTEGER LS,S,RS;
                 BEGIN...END;                  /procedure which contains transforma-
                                                tion rules (dependent on state of
                                                left neighbor, own state, and state
                                                of right neighbor)

REF (CELL)C; REAL DT;                          /DT=1 for globally synchronized
                                                L-system

PROCESS CLASS CELL(LN,RN,STATE);               /CLASS declaration for CELL

        REF (CELL) LN,RN; INTEGER STATE;       /with parameter pointers LN and RN to
                                                left and right neighbors and a STATE

        BEGIN INTEGER NEWSTATE;                /each CELL has a local variable
                                                NEWSTATE

NEXT: HOLD(DT);                                /the present CELL is suspended for a
                                                duration DT

      NEWSTATE:=TRANSFORM(LN.STATE,STATE,      /LN.STATE accesses the variable STATE
                          RN.STATE);            of the CELL with pointer LN

      HOLD(Ø);                                 /enables correct simulation of
                                                simultaneity

      IF NEWSTATE<2 THEN                       /if no division occurs then
         STATE:=NEWSTATE ELSE
      BEGIN STATE:=1;

      IF NEWSTATE=2 THEN                       /2=code for division to lengthen branch

      BEGIN C:-NEW CELL(LN,THIS CELL,1);       /C points to a newly generated cell
                                                with the LN of the present CELL as its
                                                left neighbor, and this CELL as its
                                                right neighbor, and STATE=1 (i.e.,
                                                division to lengthen the branch)

         IF LN.RN=THIS CELL THEN               /the right neighbor of the left neigh-
            LN.RN:=C;                           bor is set to the new cell C (if it
                                                was the current cell)

         ACTIVATE C;                           /C starts execution now

      END ELSE;

      ACTIVATE NEW CELL(THIS CELL,ENV,1);      /a new cell is generated as sidebranch
                                                (i.e., with this CELL as its LN and
                                                the environment as RN, and in
                                                STATE=1), and starts execution now

  END;

    GOTO NEXT;

  END;
```

Table 2 -

Sketch of discrete-event simulation of a branching pattern-generating L-system (coded in SIMULA/67)

In the case of lengthening a branch (Figure 2a) this new cell has the left neighbour of the parent cell as its left neighbour and has the other daughter cell (formed by state change out of the parent cell) as its right neighbour. Moreover, the right neigbour of the left neighbour of the parent cell is set to the new cell, and the left neighbour of the other daughter cell is also set to the new cell. In case of branching (Figure 2b) the new cell has the other daughter cell as its left neighbour and has the environment as its right neighbour (compare Figure 2). The absence of a globally accessible datastructure implies that L-systems are not explicitly spatially embedded; therefore additional conventions are needed for spatial representation.[3]

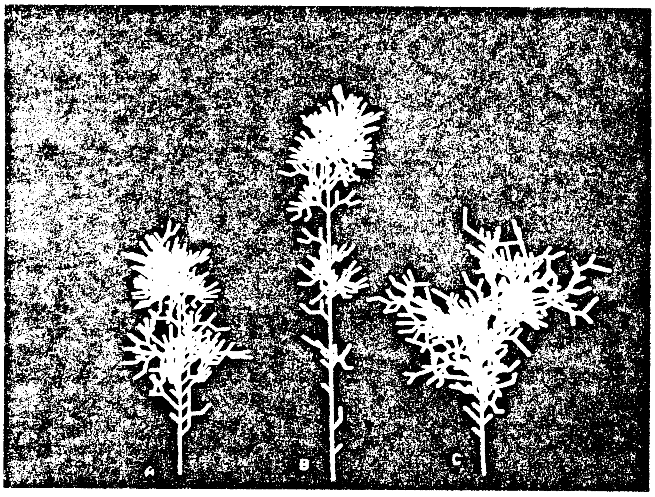


Figure 3 - Three forms generated by the same set of transformation rules but different timing regimes:
(a) Globally synchronous timing regime (L-system)
(b) Locally synchronised timing regime
(c) Partially asynchronous timing regime

Hogeweg and Hesper[3] reported on form-generation experiments with L-systems, employing the basic scheme sketched in Table 2. The forms generated by a class of very simple transformation rules are strikingly complex. As illustrated in Figure 3a, the branching structures are remarkable in their lifelike appearance. Very distinct types of forms are generated by different classes of transformation rules, and there is a marked appearance of distinct substructures (e.g., 'flowers' of Figure 2a).

We stress that we regard these models as being of *heuristic* interest for the study of plant development. By studying them we try to perceive relationships between characteristics of the forms and the properties of the transformation rules.[3,4] Since these models ignore all physiological particularities of organisms, they must be viewed as models for developing cellular forms, *not* for developing organisms. Nevertheless, to study biological development, we shall need a well-understood theory of form development for systems which share some of the properties of biological systems (e.g., cellularity). As such, the above systems are of heuristic interest for studying the development of biological forms.

Comparing the properties of L-systems with those of cellular space systems with respect to their usefulness in heuristic biological modelling, we note:

(1) Cell 'birth' and cell 'death' are directly representable in L-systems, but can be simulated only awkwardly in cell space systems when biological constraints for descendence and connectivity are imposed (see Ransom[7] for one approach).

(2) The dependence of the input neighbourhood on ancestry is an attractive feature for modelling plant development because cell-fluid contact is indeed maintained between neighbouring cells which replace a parent cell, but neighbouring cells which 'meet' later in their lifetime are not in effective contact because they will have developed too thick a cellulose wall by that time. It has not been properly recognised, however, that this property applies exclusively to plant development and not to animal development. In animals cell-to-cell communication may be established between cells not related by common ancestry, and conversely, since cells move relative to one another, common ancestry does not necessarily imply common communication links.

(3) Both L-systems and cellular space systems enforce a global simultaneity of (active) cell transformations. Such a rigic synchronous operation of cells is not a feature of biological systems. Models of biological systems which try to establish relatively simple growth rules operating on cells with very few states and very few neighbours are severely hampered by this imposed synchronicity. Asynchronous cell systems may be simulated by synchronous ones only by greatly increasing the number of states and neighbours.

ASYNCHRONOUS AND LOCALLY SYNCHRONISED CELLULAR SYSTEMS

In all cellular growth models, cells are assumed to be autonomous units and the transformation of the entire array of cells is defined by the transformations of the individual cells. However, the synchronicity of cell transformations assumed in the foregoing growth models introduces an implicit global control, which contradicts the autonomy of cells. Moreover, such global control is assumed to be absent in, e.g., biological systems, and as mentioned above, asynchronous systems can be simulated by synchronous ones only at the cost of a large increase in the number of states. Therefore it is important to investigate the effect on generated forms of relaxing the constraint of global synchronicity. For this purpose we defined asynchronous and locally synchronised versions of L-systems in such a way that the globally synchronous systems were retained as a limiting case.[5]

In *asynchronous* systems the time delays between transformations (DT in Table 2) are subject to variation from cell to cell and from one transformation to the next, as opposed to the globally synchronous case in which it was fixed and equal for all cells. A finite retention time between the determination and the execution of the transformation is maintained. Various amounts of synchronisation are obtained by varying these two time delays relative to one another. The effect of moderate asynchronisation is shown in Figure 3c. Asynchronisation leads in this case to deterioration of the distinct substructures in the branching pattern.

Asynchronous systems are the extreme opposite of the globally synchronous systems we started with. We can back up from this extreme by introducing the possibility of *local synchronisation*. In locally synchronised systems, a cell which is about to change state immediately notifies its neighbours before actually carrying out its change. Each neighbour checks to see if under the current circumstances it would also change state and, if so, immediately passes this information on to its neighbours, and so on. After such signaling is completed, all the cells involved change state simultaneously. Table 3 shows how this is expressed in SIMULA/67.

Such locally synchronised systems do not simulate globally synchronous systems faithfully; in the next time step the cell which caused the signal to stop has a neighbour on one side which has undergone one or more transformation than the neighbour on the other side. The amount of synchronisation achieved by such a regime is strongly dependent on the particular transformation rules of the system and may vary greatly from one time step to the next.[5]

A "Watch out! I am *going* to change my state" synchronisation signal contrasts with a second possible modification of the asynchronous systems in which a "Watch out! I *have* changed my state" is passed on to the neighbours after completion of the state transformation. Such systems are called *locally activated systems*. In these, a change in the cell's state causes immediate notification of the neighbouring cells, which employ the new state of the cell which notifies them in determining their next state transformation. Such locally activated systems are attractive for biological modelling in which changes in cell states, particularly if the change is cell division, may necessitate neighbouring cells' doing the same to avoid ripping of the cellular structure (Korn, personal communication).

As illustrated in Figure 3b, we found that both local regimes (synchronised and activated) tend to restore the generation of distinct substructures in the branching structures as compared to the asynchronous case.

To sum up our experiments, we found that a change in the timing regime greatly influences the shape of the individual branching patterns generated by a specific set of transformation rules. Therefore, consideration of the timing regime is obligatory when modelling the development of specific forms. However, the general properties found in previous experiments on globally synchronous systems (i.e., the striking complexity and lifelike appearance of the patterns generated by very simple rules, and the distinctness of the different forms and of the substructures) are retained in the asynchronous variants. This is true to a greater extent for locally synchronised and locally activated systems than for the partially asynchronous systems. This was to be expected because the transformations of the cells are locally determined and therefore more influenced by local synchronisation than by partial synchronisation of remote cells.

DISCUSSION

The morphology of organisms may seem to us marvellously complex. One of the central questions of biology (and for that matter, of science in general)

```
PROCESS CLASS CELL (LN,RN,STATE);
        REF (CELL) L.,RN; INTEGER STATE;
        BEGIN INTEGER NEWSTATE;                /identical to CELL of Table 2

        REF (HEAD) BRANCHES;                   /pointer to list of sidebranches

        REF (LINKAGE) BR;                      /pointer to sidebranch

        BRANCHES:-NEW HEAD                     /list for sidebranches is initiated

NEXT: DT:=NORMAL(MEAN VARIANCE,U);             /DT is computed

      IF DT>Ø THEN HOLD(DT);

      NEWSTATE:=TRANSFORM(LN.STATE,STATE,      /as in Table 2
                          RN.STATE);

      IF NEWSTATE=STATE THEN GOTO NOINT;

      IF STATE=1 AND NEWSTATE>1 THEN GOTO      /synchronization signal is not passed
      NOINT;                                    on if the STATE does not change

      IF RN=/=ENV AND RN.EVTIME>TIME THEN      /check against cyclic reactivation;
                                                no reactivation if the cell is
                                                already scheduled for this exact time

        REACTIVATE RN DELAY Ø PRIOR;           /the right neighbor is reactivated
                                                with priority

      IF LN=/=ENV AND LN.EVTIME TIME THEN

        REACTIVATE LN DELAY Ø PRIOR;           /same for left neighbor

      BR:BRANCHES;                             /BR points to beginning of list

      FOR BR:-BR.PRED WHILE BR=/=NONE DO
      INSPECT BR WHEN CELL DO
      IF EVTIME>TIME THEN

        REACTIVATE THIS CELL DELAY Ø PRIOR /all sidebranch CELL in contact are
                                            reactivated

NOINT: HOLD(Ø);                               /delay to enable parallel processing

                  < continued like CELL in Table 2 >
```

Table 3-
Cell definition for branching patterns generated by locally synchronized systems (coded in SIMULA/67)

is how to decompose such complexity into simple interacting components such that these components generate the perceived complexity. The models discussed in this paper address this general question by studying the forms generated by certain simple interacting components which satisfy some of the constraints that seem reasonable for organisms. That is not to say that any of these forms exist in nature. However, the successive generalisations starting with the classical cellular space systems and going from them to L-systems and then to locally synchronised cell-development systems incorporate progressively fewer properties which are unreasonable for biological systems. The cell structures obtained using these generalisations have progressively increasing heuristic value for understanding biological structures, even though the models contain little or no biological information about particular mechanisms or particular organisms. That was not their purpose.

The generalisation of L-systems to partially asynchronous, locally synchronised, and locally activated systems was caused by the implementation of a simulation system for L-systems in the heterarchical (as opposed to hierarchical) discrete-event simulation medium provided by SIMULA/67. This simulation system was initially meant to be merely a programming exercise, but it showed us that cells could be explicitly programmed as autonomous units in this medium, and it made apparent the artificiality of globally synchronised cell transformations. In contrast, cellular spaces, L-systems, and discrete time-step systems generally force the cells to be conceived of as 'slaved' parts of the whole and are therefore implicitly holistic. Uncovering this implicit holistic concept in models employing synchronous operation is an important finding because cellular models are often used as examples to show that simple *local* control may result in a seemingly complex *global* phenomenon.

In my view a primary function of simulation modelling is to help the modeller convert his vague ideas of what the model should look like into an explicit unambiguous form: a model expressed as a computer program. The medium in which the model is expressed is of crucial importance because it guides the initial formulation of the model and determines which extensions are conceptually 'easy.' The medium to express simulation models is commonly called language, but its essential features are not so much those which computer languages share with natural languages, but rather the control and datastructures the computer language provides. With respect to control and datastructures, natural languages do not differ very much, but computer languages do. Therefore different simulation languages provide different thinking media, which profoundly influence our ways of thinking and our choice of models.

ACKNOWLEDGEMENTS

My ideas about auxology and about simulation have taken form in discussions with Dr. B. Hesper.

Dr. B.P. Zeigler encouraged me to proceed along the line of asynchronisation of cellular systems and suggested local synchronisation as a mechanism to investigate.

REFERENCES

1 GARDNER, K.
The Fantastic Combination of John Conway's New Solitary Game "Life"
Scientific American October 1970 pp. 120-123

2 HERMAN, G.T. LIU, W.H.
The Daughter of CELIA, the French Flag, and the Firing Squad
Simulation vol. 21 no. 2 August 1973
pp. 33-41

3 HOGEWEG, P. HESPER, B.
A Model Study on Biomorphological Description
Pattern Recognition vol. 6 1974 pp. 165-179

4 HOGEWEG, P.
Topics in Biological Pattern Analysis
PhD dissertation Bioinformatica University of Utrecht the Netherlands 1976

5 HOGEWEG, P.
Locally Synchronized Developmental Systems: Conceptual Advantages of Discrete-Event Formalism
In B.P. Zeigler, editor, *Frontiers in Systems Modelling* (in press)

6 LINDENMAYER, A.
Mathematical Models for Cellular Interactions in Development. I: Filaments with One-Sided Input. II: Simple and Branching Filaments with Two-Sided Inputs
Journal of Theoretical Biology vol. 18 1968
pp. 280-312

7 RANSOM, R.
A Computer Model of Cell Growth
Simulation vol. 28 no. 2 February 1977
center insert

8 SOHNLE, R.C. TARTAR, J. SAMPSON, J.R.
Requirements for Interactive Simulation Systems
Simulation vol. 20 no. 5 May 1973
pp. 145-151

9 ULAM, S.M.
On Some Mathematical Problems Connected with Patterns of Growth of Figures
Reprinted in Burks, editor, *Essays in Cellular Automata* (University of Illinois Press, Urbana, 1970)

10 von NEUMANN, J.
Theory of Self-Reproducing Automata
University of Illinois Press Urbana 1960

11 WINOGRAD, T.
A Simple Algorithm for Self-Replication
MIT MAC Memo 198 1970

12 ZEIGLER, B.P.
Theory of Modelling and Simulation
Wiley New York 1976