LCAU.DOC

Harold V. McIntosh
Departamento de Aplicación de Microcomputadoras,
Instituto de Ciencias, Universidad Autónoma de Puebla,
Apartado postal 461, 72000 Puebla, Puebla, México.
E-mail:mcintosh@servidor.unam.mx

April 8, 1988
revised October 12, 1992

Abstract

Previous collections of cellular automata programs have grown, both as the analysis of new aspects of their behavior has continually been included, and as the range of parameters of the automata has increased. During all this time, the programs have lacked an operation manual, which is the substance of the present document. Previously, it has been necessary to rely on the source code, which has always been made available along with programs, but whose commentary has never been expository. A set of lecture notes, *Linear Cellular Automata* accompanies the programs, and explains most of their features.

The first collection, LCA, covered a small range of radii and state numbers, with supplementary programs to calculate cycles and de Bruijn diagrams. Most of the rules were required to be totalistic.

The second collection, LCAU, contained three principal extensions:

1. General rules, not just totalistic rules, could be analyzed.

2. Cell densities and block probabilities were calculated according to mean field theory and local structure theory using options in a new submenu.

3. The option to produce de Bruijn diagrams was incorporated in a submenu.

Consequently a more detailed and convenient analysis of (k,1), k=2,3,4 and (2,2) automata could be made than previously.

The current revision, LCAU (second series), extends the range of the parameters k and l considerably, including half-integer values for r. An entirely new submenu, for the calculation of ancestors, has been introduced, based on de Bruijn diagrams. Configurations deduced from the de Bruijn diagrams can now be transferred back to the main menu, so that their full screen evolution can be observed.

The programming language REC has been included, whose principal application lies in organizing demonstrations, but it will eventually allow storing and retrieving demonstrations from disk.

Finally, extensive provision has been made for random and partially random connections of the cells. To study a related phenomenon, the Chaté-Manneville effect, higher dimensional homologues of many of the automata can be studied.

Some slight provision has been made for a mouse.

Contents

1 Introduction

As often happens with program collections, LCAU has grown from year to year; three stages of development can probably be distinguished. The first began with an article in Byte magazine which described a four state automaton suitable for use with microcomputers having a color monitor.

As the range of automata covered and the number of characteristics analyzed increased, a more elaborate manual was prepared and the structure of the individual programs was given a more elaborate organization. The collection remained fairly constant for a while, but within the past year several new facilities have been added. These include the addition of disk storage, the incorporation of the REC programming language, and the provisions which were made to work with automata of the Chaté-Manneville class.

1.1 LCA.C

The disk LCA.C was released in the summer of 1987, 9 programs written in "C" for calculating and displaying the evolution of linear cellular automata. Both source code and a separate disk of EXE files are provided. Although the programs represented a point in the evolution of the course Fortran III offered regularly during several preceding semesters at the Universidad Autónoma de Puebla, and were promoted at the XVI Feria de Puebla, they had their principal inspiration in an article in the December, 1986, issue of Byte magazine which explained how cellular automata could lead to intricate and complicated designs, with a certain aesthetic appeal.

Kenneth E. Perry,
Abstract Mathematical art,
Byte, December 1986, pages 181-192

The programs in the LCAKR.C set ran through the k-range of 2, 3, and 4 states per cell, as well as interactions between first, second, or third neighbors (the r-range). The combinations $(4, 1)$ and $(4, 2)$ are surely the most colorful, but the binary first neighbor version $(2, 1)$ is more amenable to an exhaustive analysis.

1.2 LCAU.C

Programs in the second series were named LCAU to distinguish them from members of the first series. In the original series, in all cases except for $(2, 1)$, only totalistic rules were considered. Totalistic rules are those for which the transition depends only on the number of cells of different kinds in each neighborhood, but not on their precise arrangement. More exactly, each state gets assigned an integer in the range 0 to $k - 1$ as a weight, the actual transition being a function of the weighted sum of all the neighbors (including the evolving cell). The artistic effects arise from assigning colors to each of cell values.

1.2.1 General rules

Although the use of totalistic rules serves to reduce the enormous number of possible automata greatly, it excludes many interesting possibilities arising from a more detailed specification of the evolutionary rules. When k, the number of states per cell, is small, there is not too much which is possible in the way of design, but once it reaches 4 or beyond some interesting constructions become possible.

1.2.2 Demonstrations

Several interesting demonstrations become possible at the level of a $(4, 1)$ automaton with a general rule.

Binary counter LCAU41.C in particular, contains enhanced rule and line editing menus which allow experimentation with the design of rules. Certain of the demonstrations in LCAU41 show how the design process can be used. There is an example of a "binary counter" which consists of a glider gun together with bistable targets. In one state the target absorbs the glider and changes to the other state. In the second state the target passes the glider, reverting to the first state. Thus half the gliders are lost to each target, whose state forms a binary counter whose carry is represented by the gliders which are allowed to pass.

Glider gluon demonstration Another demonstration shows how a bouncing glider may shuttle between two obstacles - still lifes - drawing them ever closer together. Just as the glider is about to be crushed, the walls coalesce but the glider escapes to one side, seeking new walls to conquer. Multiple gliders may go about their business, competing for the wall if they lie on opposite sides or hastening to squeeze if they lie on the same side.

Jumping bean A third demonstration shows a slow glider propelled by an internal glider or gliders bouncing back and forth - a sort of Mexican jumping bean. When a fast glider overtakes a slower glider, they coalesce, producing an even slower glider.

Gliders of two velocities A fourth demonstration shows gliders of two different velocities, which can be used to set up a remote still life whose reaction to further gliders gives it a life of its own.

Such constructions can be used to generate interesting patterns, but they also serve theoretical ends as well. For example, the binary counters establish the existence of structures with both exponentially long transients and exponentially long cycles. Since they still use several cells to establish the basic components and their spacings, they still do not reach theoretical maxima; but they do lie within certain factors of such maxima. When k becomes larger still, universal Turing machines can be programmed, but this probably requires a k of at least 6 or 8.

1.2.3 Block probabilities

Another extensive addition which has been made to the programs is to be found in the series PROB.C, which can be invoked by typing `t` in the main menu (the old totalistic rule number can still be utilized by typing `T`); in fact their inclusion more than doubles the size of the programs. These new programs permit a statistical survey of the properties of the automaton.

Originally they calculated simple probabilities on the basis of ideas which go by the name of "mean field theory," whose results were plausible but not entirely convincing. At about the time the programs were being updated, two interesting articles appeared in the literature:

> W. John Wilbur, David J. Lipman and Shihab A. Shamma,
> On the prediction of local patterns in cellular automata,
> Physica 19D 397-410 (1986).
>
> Howard A. Gutowitz, Jonathan D. Victor and Bruce W. Knight,
> Local structure theory for cellular automata,
> Physica 28D 18-48 (1987).

These articles, from differing points of view, showed how to take correlations between cells into account by calculating the probabilities of strings of cells. Rather than taking individual probabilities as fundamental and deducing the probabilities of combinations, the process is inverted; self consistent probabilities for strings (or blocks) of a certain length are found from which the probabilities of individual cells are obtained by an averaging process.

The calculations of these articles were then included among the options of the `t` submenu, so that probabilities derived from blocks of length up to 6 could be compared with simpler estimates and with the actual performance of the automaton.

1.2.4 de Bruijn diagrams

A third feature of the second series was the incorporation of the de Bruijn diagrams within the main program, together with a visual representation in terms of chords of a circle. It was still not possible to transfer the cycles obtained back to the main program without copying them on paper and editing the initial line with the option `l`.

Limitations of space and time severely restrict the lengths of periods which can be analyzed. Although interesting gliders and cycles are found within the accessible range of the program, there are many others of longer periods which manifest themselves experimentally when the evolutions are run. It would be nice if the exhaustive analysis afforded by the de Bruijn diagrams were feasible for the longer periods that show up on the screen, but they would really require a faster computer, more memory, and probably programs incorporating finer details of the algorithms used.

The programs contained a bare minimum of help facilities, in the sense that menus of one type or another were presented at various stages in the evolution of the programs, and others are sometimes available by typing a question mark, just as a slash often cleared portions of the screen.

A manual consisting of the lecture notes for Fortran III is in preparation, for which chapters are planned describing by accompanying programs. As is well known, the preparation of manuals always lags the evolution of the programs which they are supposed to describe.

There are still some interesting problems of presentation — recall that Fortran III is supposed to be dedicated to graphical techniques. Presentation of simple evolution is easily solved, since the resolution and velocity of the graphics controllers included as standard equipment in IBM PC's and their clones is adequate. Unfortunately color monitors and their controllers are sometimes seen as premium equipment which was not included in a given installation, so that a monochromatic rendering of the color displays must be endured.

Even so, the speed and screen resolution which is available in the present generation of equipment is only marginally satisfactory, having only a fraction of the resolution of pen and ink plotters which have been commonly available. Once statistics pertaining to the evolution of automata are to be presented, it is found that there are many more parameters than are comfortable, which leads to the use of shading, complex surface representations, even stereographic view. It is in this area that some interesting results can be obtained, but mostly ones which whet one's appetite for the next generation of equipment.

Likewise the use of the de Bruijn diagram and the reduced evolution diagram, even without their probabilistic versions,requires line drawings of a complexity which quickly surpasses the capability of the present generation of graphical displays. Although the complexity of these diagrams increases exponentially — making even modest values of parameters permanently inaccessible; still, even moderately better graphics equipment will permit an instructive display of the simplest cases.

Although the menus vary slightly from program to program, they generally conform to a uniform pattern, whose constituents are described below.

1.3 LCAU.C: SECOND SERIES

The third stage, LCAU: SECOND SERIES, incorporates still more major improvements.

1.3.1 Ancestors, Garden of Eden

For those automata where the space and time requirements are not excessive, an entirely new submenu has been included, which provides for the calculation of ancestors, and the detection and calculation of Gardens of Eden. Implicit in the lack of a Garden of Eden is reversibility, which can also be checked.

1.3.2 REC programs

The difficulty of preparing good demonstrations (Recall the presentation of the LCA programs at the Feria de Puebla) motivated the incorporation of a REC compiler in the programs, with its menu of operations. In turn this permitted a reorganization of the scheme for incorporating sample rules in the programs, and the provision for reading collections of prepared rules from disk.

1.3.3 Randomly connected cells

Just before the involvement with the Chaté-Manneville automata, the programs had been modified to handle a certain number of random connections between the cells, in addition to the perfectly regular connections arising from the lattice structure. Indeed, it was originally conjectured that the collective behavior of these automata was due to an increasing randomness in the evolution function with increasing dimension; but that proved not to be the most important effect.

1.3.4 Chaté-Manneville automata

The most extensive part of the third revision was precipitated by the article

> H. Chaté and P. Manneville
> Evidence of Collective Behavior in Cellular Automata,
> Europhysics Letters 14 409-413 (1991),

and several events following its publication. The principal item of interest is a cyclic overall behavior of the probability densities as one of these automata evolves; the automata themselves are multidimensional.

Automata of different dimensionalities can have neighborhoods of the same size, depending on the number of ways that the size can be factored. Such differences have to affect the correlations between different cells of the neighborhood when calculating probabilities, providing a motivation for using different neighborhoods within the same program. Their adjunction of a program set emphasizing one-dimensional automata was due to the elaborate facilities for displaying probabilities which already existed.

1.3.5 Mouse buttons

There is next to no necessity at all to use a mouse in any of the LCAU programs, but there are one or two occasions in which there is a movable cursor present. Either the four keyboard arrows, or some of the older equivalents such as space and backspace, can be used to move the cursor.

No mouse drivers are included in LCAU, but there are many "external" mice (mouses?) on the market which work by inserting characters into the input buffer of the computer to which they are attached.

Such a mouse can be programmed to create arrows corresponding to its movements, CONTROL ARROWS for the two buttons, and ESCAPE for a double button press. It will then be compatible with the option assignments in LCAU.

2 Interdependence of submenus

Given that each LCAU program can calculate many quantities related to the performance of an automaton, the general programs have been organized into submenus, each of which is described in detail in its own section.

Figure 1 shows the interrelationship of all these submenus, most of which are adjuncts of the main menu.

The initial screen in all programs bears a copyright notice and a very short description of the program. While the inexperienced user is busy reading the screen, a random number generator is wasting time, so that there will usually be a different display every time the program is run; likewise a different sequence of random rules and initial lines. No effort has been made to see how much this initial idling degenerates the performance of the random number generator.

As a general practice, return from all submenus, at every level but the highest, results from typing a carriage return. The top level is treated as an exception to prevent the overenthusiastic use of CARRIAGE RETURN. An unintended exit to DOS would result in the loss of several option settings; to avoid this inconvenience, the program must be terminated via the option `q`. It is also possible to quit by typing `n` in response to the query in the evolution panel.

Although considerable effort has been expended to keep the notation uniform throughout all the LCAU programs, and throughout all the submenus within an individual program, there are exceptions. This is almost unavoidable if a large number of options exist, all of them designated by single letters.

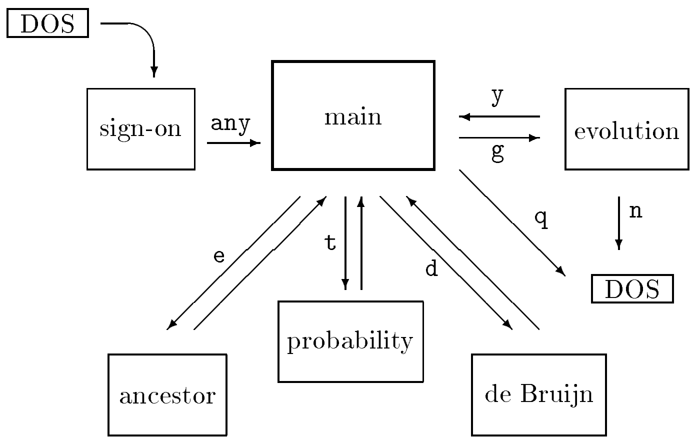

Figure 1: Interdependence of LCAU submenus. All unlabeled returns are generated by typing CARRIAGE RETURN.

3 Copyright notice

The user gets only one chance at the copyright notice, mainly because it hides a random number generator which ensures that different demonstrations will be proffered every time the program is executed.

Whereas the authors reserve commercial rights to the programs, they place no restriction on copying for private use or study, save for insisting that the programs be copied in their entity, without any alterations whatsoever.

Naturally the availability of the source code will allow interested persons to correct errors or adapt the programs to their own requirements; general usage suggests that the code be annotated accordingly, and treated with the same respect as the original. The authors will, of course, appreciate being informed of errors, suggestions for improvement, or ideas concerning the coverage of additional topics.

The apparent conflict between “no alterations” and “make your own corrections” is easily resolved. Sometimes the only change made to a program is to remove or to change

```
-------------------------------------------------------

 Copyright (C) 1987, 1988 H.V.McIntosh, G. Cisneros S.
           *** LIFE in One Dimension ***
 Three States - Black(0), Cyan(1), Magenta(2).
 First neighbors - one on each side, three altogether.
 Complete transition rule - random, edited, or stored.
 Totalistic transition rule - random, edited, or stored.
 Initial Cellular Array - random, edited, or patterned.
 Some rules are fragile and require several tries before
 manifesting and interesting evolutionary pattern.
 Use any key to terminate a display in progress.
 now, press any key to continue.....
 -------------------------------------------------------
```

Figure 2: copyright notice.

the attributions concerning its origin; this is not considered fair, and should be avoided when reproducing *any* program.

4 Main menu

The Main Menu Figure 3 generally offers the following selection, whose details vary slightly from one automaton to another:

`r` - edit the general rule
`l` - edit the line
`g` - graph the evolution
`x` - generate a random rule
`y` - generate a random line
`u` - generate a sparse line
`T` - edit a totalistic rule
`S` - edit a semitotalistic totalistic rule
`#` - execute stored rule number nn
`@`nn - execute totalistic rule number nn
`$` - execute 12 totalistic rules starting with number nn
`w`nn - execute Wolfram rule number nn (LCAU21 only)
`t` - enter probabilistic submenu
`d` - enter de Bruijn submenu
`q` - quit

Most of the options lead into further submenus; some of which still deal with information visible within the main menu, such as `r`, which edits the rule, or `l`, which edits the line. Others invoke full screen displays of their own, notably the option `g`, which

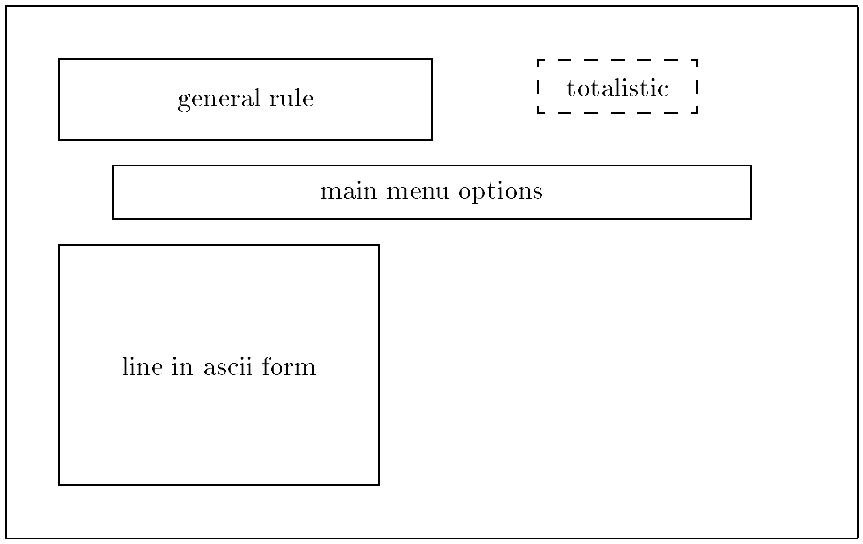

Figure 3: main menu, showing rule panel, option list, and line panel.

displays the evolution of the automaton line by line. The interrelationship of the different full-screen displays is shown in Figure 1.

Each of these options is described in further detail below.

4.1 Rule editing

Insofar as space permits, a panel for the general rule is always displayed in the upper left hand corner of the main menu. Selecting the option `r` will place the cursor on the bottom line of the rule panel, where the values of the transition function may be entered by typing the appropriate digit. The cursor may be positioned by space, backspace, and the horizontal arrows.

Some programs have tabs, and movement to the ends of the line; others do not. Likewise, the neighborhood may be marked and unmarked in some programs — those which allow extensive line editing, for the most part.

Space for totalistic rules or semitotalistic rules is allocated in various portions of the main menu, according to space requirements and competing information which may have to be displayed.

4.1.1 Full rules

To edit a rule, either general or totalistic, one may move the cursor and insert values for the cell above the cursor. Some programs have a more elaborate cursor, with tabs, wraparound, and the possibility of flagging values which will not be altered by using the random rule generator `x` (`X` overrides the flags). Again these features are experimental, and may possibly be confirmed in future versions of the programs. Later on the special commands which apply to LCAU41 are shown.

4.1.2 Totalistic rules

Totalistic rules are those for which the transition depends only upon the sum of the states of the cells in the neighborhood, but not upon their arrangement. Only a small fraction of the possible rules have this form, nevertheless just that reduction makes it possible to select rules for very large neighborhoods or large state sets. Otherwise too many transitions have to be specified to fit on the screen, or for their definition to be conveniently remembered.

Depending on the space available, the totalistic rule will either appear in the upper left hand corner of the main menu, or in the right central portion, to one side of the line panel. Cursor movements and state definitions are available in all such mini-menus; other options vary from automaton to automaton.

4.1.3 Semitotalistic rules

A slight generalization of totalistic rules is to make the transition depend explicitly upon the state of the central cell, but otherwise only on the state sum of the remaining cells; the combination is called a semitotalistic rule. Neighborhoods of even length do not have a central cell, so the concept may be extended still further to include partitioning the cells into two groups, then using the pair of state sums to define the transition.

The placement of the semitotalistic rule follows the same general principles as that of the totalistic rule, the same also to be said about the variety of options.

4.2 Line editing

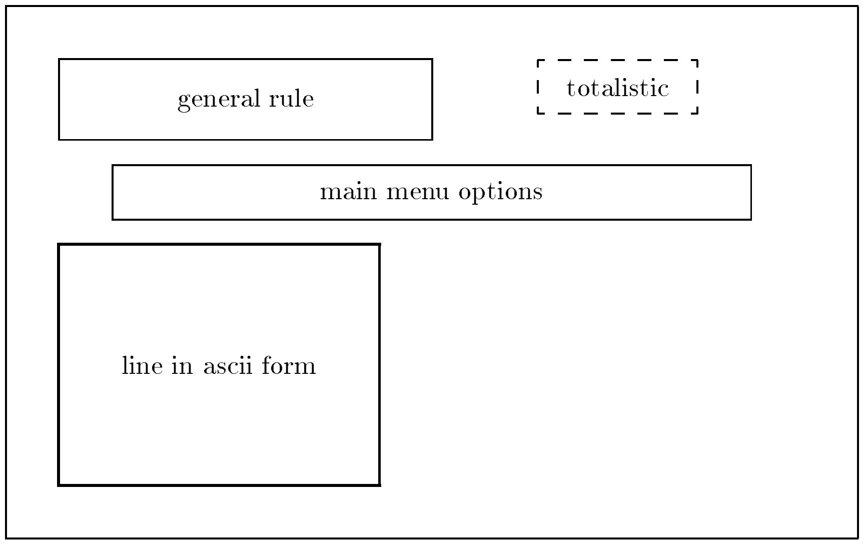

Figure 4: line editing menu. The line editor works on the line of cells, which is divided into 8 lines of 40 cells.

4.2.1 Line editing menu

(Figure 4) - Lines are edited by essentially the same procedures that the rules are, but the long line of 320 cells is broken down into 8 lines 40 cells, to make movement across the line using the up and down arrows more rapid. LCAU41, which corresponds to one of the simplest automata for which rules may be designed to order, has many additional line editing options, all experimental, which can be used to facilitate the design. No doubt more will be added before the existence of all of them is officially announced.

`0,1,2,3, ...` - enter state
MOD- move cursor downward
MOU- move cursor upward
MOR- move cursor forward
MOL- move cursor backward
`<` - move cursor to left margin
`>` - move cursor to right margin
`z` - clear single row
`Z` - clear full array
ˆ - repeat the line just above the cursor's line
`x` - unmark transitions
`q` - clean up the area
`=` - insert transition
`+` - set marker for transition in auxrule
`-` - remove marker for transition from auxrule
`*` - insert transitions for the whole line
`.` - point evolution
`?` - evolution of the whole line
`/` - conditional evolution
`c` - test consistency everywhere
`C` - test consistency for marked neighborhoods
`m` - try to match a given evolution

5 Full screen evolution

Figure 5 shows a typical screen of evolution. Evolutionary screens may be interrupted at any line by pressing any key; alternatives will then be offered to continue the display (CARRIAGE RETURN), exit the program (`n`), or return to the main menu (nominally `y`, but it is the default). It is a general principle in all parts of the program, that they keystroke which interrupts any activity is discarded. Therefore a new activity cannot be initiated by simply typing its letter, although typing it twice in succession will usually work.

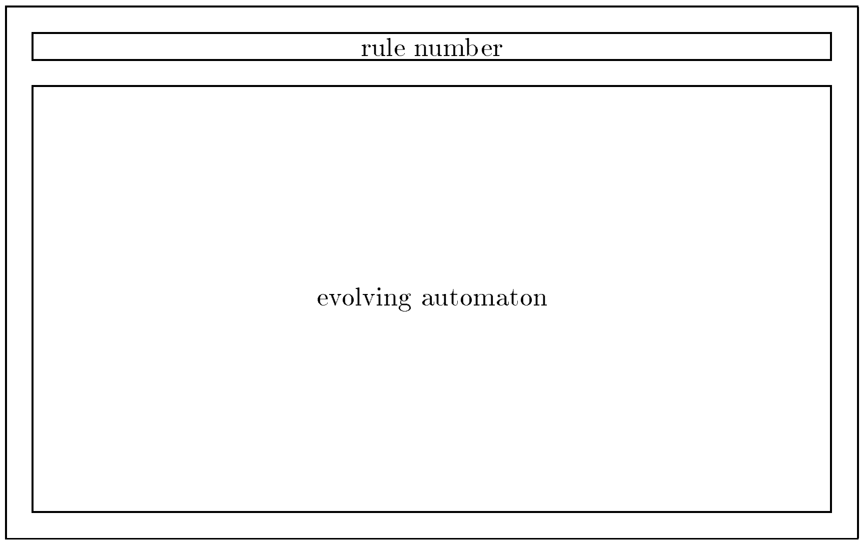

Figure 5: A screen of evolution.

6 Probabilistic menu

The probabilistic menu contains all the options pertaining to the statistics of automaton evolution. The greatest variation between programs results form differences in graphical treatment of different sets of probabilities.

For binary automata, only one probability is required, since the probability of the other state is the complement of the first. Consequently, simple two- dimensional graphs suffice to show how probability changes from one generation to another; for instance through mean field curves. Binary automata are also simple enough that contour maps can represent block probabilities for pairs.

Trinary automata have two independent probabilities, for which contours in triangular coordinates suffice; beyond three states simplices of at least three dimensions are required.

Stereopairs showing sections through a pyramid can be used to display the probabilities of quaternary automata. Since a certain amount of practice and training are necessary to visualize stereopairs, or else special optical equipment, many people fail to appreciate stereopairs. Other representations are possible, including dynamic displays with rotation, but they do not form part of the LCAU program set.

Beyond four states it becomes increasingly difficult to represent probabilities, although most of the programs contain vestiges of the quaternary treatment. These are due to a certain laziness in programming, and will gradually be eliminated from the programs.

6.1 Binary automata

The layout of the probability menu varies with the number of states; to a slight extent the options also vary, according to the necessities of the presentation. For binary automata, the layout is shown in Figure 6.

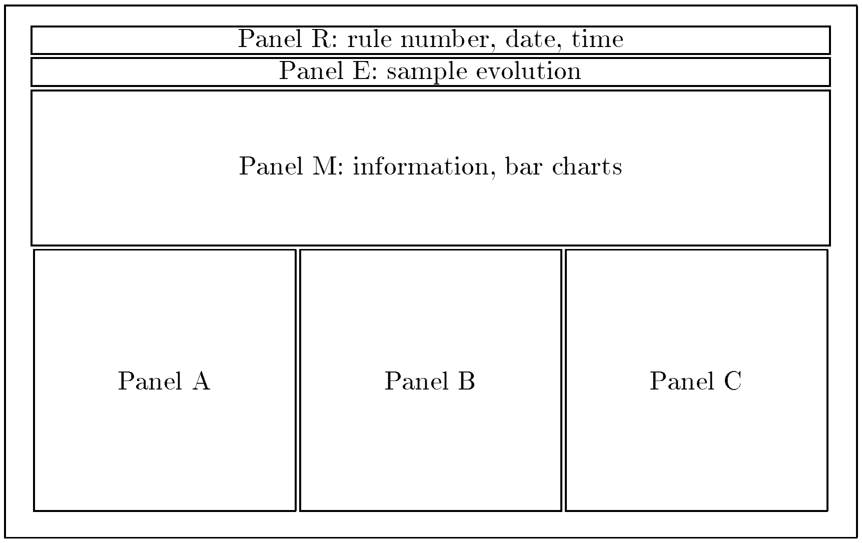
Panel R: rule number, date, time
Panel E: sample evolution
Panel M: information, bar charts
Panel A
Panel B
Panel C

Figure 6: A typical screen within binary probabilistic submenus, showing the placement of the different options. Typing `?` will summon the menu.

By typing `t` in the main menu, one arrives at a separate display, implemented in the programs PROB.C, which will perform several statistical analyses of their automata. The programs vary considerably with the number of states, since they attempt to represent the relative probabilities as points within a simplex. For two states, the results are trivial, for three states the diagrams are planar and interesting, for four states the graphical capabilities of the screen are strained; for five and beyond some other representation would be required.

The generic options are:

`a` - a priori estimates
`m` - evolution of cells and pairs
`M` - 50 generation evolution of cells and pairs
`g` - 50 generation evolution of single cells
`G` - 200 generation evolution of single cells
`s` - graph probabilities in stereo
`t` - graph probabilities, show contours in simplex
`1` - 1-block local structure theory iterations
`2` - 2-block local structure theory iterations
`3` - 3-block local structure theory iterations
`4` - 4-block local structure theory iterations
`5` - 5-block local structure theory iterations
`6` - 6-block local structure theory iterations
`+` - select blue-cyan-white
`-` - select red-green-yellow

`e` - show 16 lines of evolution
`/` - clear screen
`?` - show menu
carriage return - exit

Options `5` or `6` may not be available if they require too much time or space, `t` shows two-block probabilities for $k = 2$ automata, and there may be variants on `s`. For $k = 2$, the commands `x`, `y`, `z`, `w`, `v`, `i`, and `j` produce graphs for self-consistent 1-block probabilities for varying numbers of generations and numbers of iterations.

6.2 Trinary automata

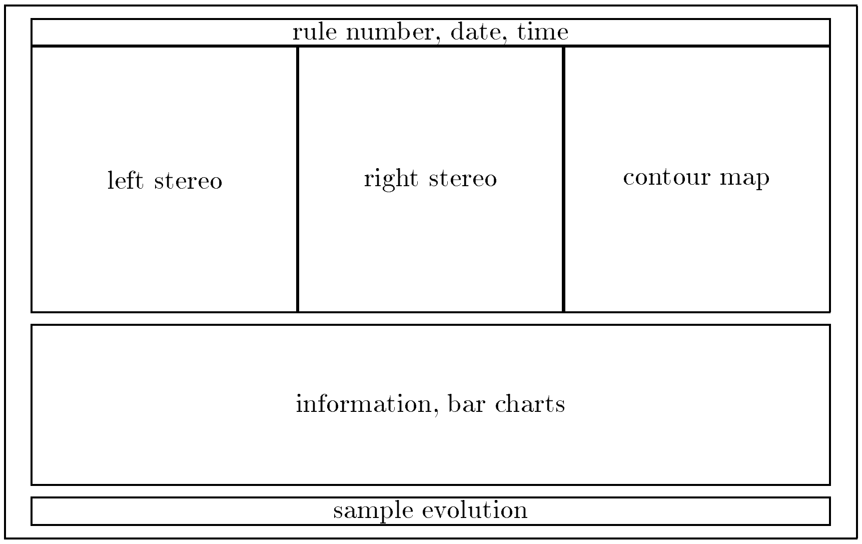

Figure 7: A typical screen within trinary probabilistic submenus, showing the placement of the different options. Typing `?` will summon the menu.

The principal difference between the binary and the trinary probability layouts consists in moving the sample evolution and bar charts to the bottom of the screen, to leave space for a trilinear diagram at the top. In particular, Panels A, B, and C no longer exist, although there is a similar apportionment of space. For most purposes only the rightmost panel is used, to hold a contour map of the vector difference in probability between the two successive generations.

Stereopairs representing this difference can be shown in the other two panels, one for the left image, the other for the right image, which are designed for crosseyed viewing.

It is also possible to view the differences of probabilities of the individual states.

6.3 Quaternary automata

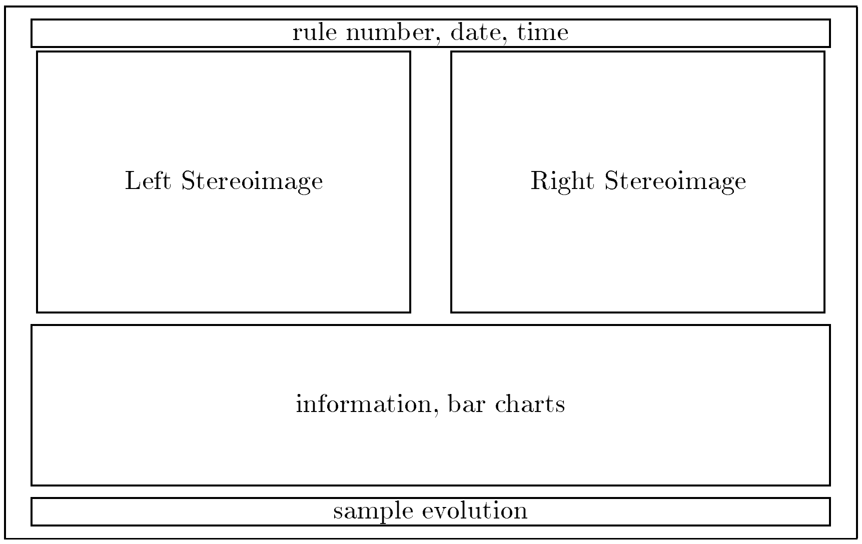

Figure 8: A typical screen within the quaternary probabilistic submenu, showing the placement of the different options. Typing `?` will summon the menu.

The quaternary probability layout is similar to the trinary layout, with the noticeable difference that a large stereopair occupies the top half of the screen.

The probability contour map is three dimensional, so it is represented by plane sections through the simplex, or unit tetrahedron. The option `u` selects one family of planes, the option `v` selects a family whose members interpolate the first; between the two one can locate prospective minima more accurately.

Likewise, the tetrahedron may be represented in two ways. The first, sitting on its base, which is the default selection, may be chosen through the option `O`. The second alternative is to think of the tetrahedron suspended between a pair of opposite, mutually orthogonal edges; `E` gives the corresponding presentation.

6.4 More than four states

Beyond four states the layout for the probability panels is very poorly developed, and generally misleading. This state of affairs is a candidate for correction in future editions of the programs.

6.5 More than one dimension

There are interesting coincidences of neighborhood size between one dimensional automata and various multidimensional automata; mostly such coincidences occur for automata of fairly large radii. Those combinations for which dual, treble, or higher dimensionality is appropriate have their own lists of operators, most all of which are to be found in the probability submenu.

The LCAUKR programs with $K = 2$ and R in the range from 3 to 6 are primarily designed to explore automata of the kind described in the article of Chaté and Manneville, previously cited.

These automata are multidimensional, so each LCAU within the relevant range has been modified to include several multidimensional neighborhoods having the same volume as the corresponding one dimensional automaton. Also, for better statistics, an array with 16K cells has been included, even though this interferes with the de Bruijn option `d` and with editing the cell array `l`.

The only effective submenu for this subset of LCAU programs is `t`, the probability option. However, the main menu is still required to define the rule; to do that is so cumbersome that only totalistic `T` or semitotalistic `S` definitions are practical. Random lines `y` and random rules `x` can still be generated, and the evolution can still be viewed line by line `g`. The ancestor option `e` was never implemented for these automata.

Exit from almost everything, especially the submenus, is via CARRIAGE RETURN. To avoid inadvertently abandoning the main program, its own termination lies with `q`, as always.

The function key `f1` will produce a list of demonstrations, which is essentially nonexistent in these programs. However, `f2` will produce a list of REC demonstrations from which a selection can be made by moving the cursor. These, although not numerous, are designed to show off the Chaté-Manneville automata, otherwise describable as "electronic Swiss cheeses."

The REC program for the selected demonstration may be displayed and edited using the key `f3`. Within the `f3` option, keys `f1` and `f2` will display panels giving a concise summary of REC and its syntax, but this should have been learned separately by having previously read the appropriate documentation files. Page-up and Page-down will scroll through all the valid REC symbols; likewise moving the editor's cursor with arrows or an external mouse (which has been programmed to transmit the arrows) will gloss the designated symbol.

REC programs are to be edited from the main menu, but executed from the probability submenu; this is purely a consequence of changing between the screen modes which are most convenient for the two activities. A similar remark applies to the way that the rule number gets defined.

7 The de Bruijn menu

The de Bruijn menus share one characteristic with the probability menus, in that their layout varies somewhat according to the number of states. Although the number of vertices in the de Bruijn diagram is affected by the number of states as well as the size of the neighborhoods, limits on the range of parameters caused by constraints on the amount of storage space and length of running time are the main reasons for this variability.

There are two kinds of de Bruijn diagrams that can be computed - those showing the counterimages of a uniform string,and those which isolate strings satisfying a certain combination of shifting and periodicity. Letters are assigned to them consecutively, but the combination of period and displacement varies widely because of the differing number of combinations that are possible.

Figure 9 shows the general format of the screen according to the de Bruijn submenu; the arrangement is slightly different for $(2, 1)$ automata because of the large number of

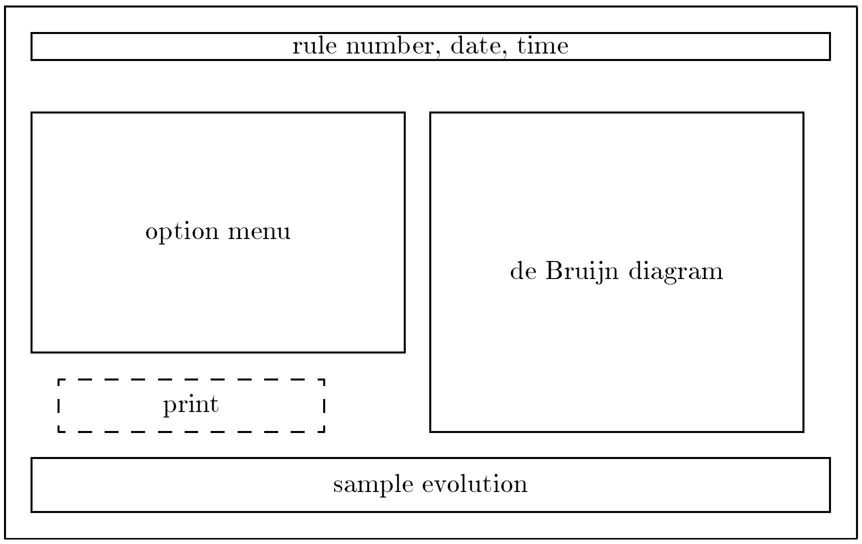

Figure 9: de Bruijn menu

combinations which can still be calculated within modest limitations of space and time. But in all cases, typing `?` will cause the option menu to appear, even after it has been overwritten during the execution of one of the options.

7.1 (2,1) de Bruijn diagrams

The parameters governing the shift-period de Bruijn diagrams are the number of generations involved and the shifting required; there are usually enough letters in the alphabet to assign each combination a unique letter. An exception is the case of binary automata with first neighbors. The layout in Table 1 shows how to specify combinations of period and shift.

a	b	c	d	e	f	g	h	i	j	k	l	m
.	.	.	.	.	.	a	.	.	.	.	.	.
.	.	.	.	.	+	b	+	.	.	.	.	.
.	.	.	.	+	+	c	+	+	.	.	.	.
.	.	.	+	+	+	d	+	+	+	.	.	.
.	.	+	+	+	+	e	+	+	+	+	.	.
.	+	+	+	+	+	f	+	+	+	+	+	.
+	+	+	+	+	+	g	+	+	+	+	+	+

Table 1: Binary de Bruijn menu layout.

First type a letter in the range `a` to `m`, for a shift lying between 6 cells to the left and 6 cells to the right. Then type a second letter in the range `a` to `g`, for the periodicity. Thus, `ga` will yield still lifes, `ch` designs which shift right one cell every three generations, and so on.

The combinations marked with a `.` are superluminal patterns, the remainder travel at the velocity of light or less, and are marked by `+`.

7.2 Other de Bruijn diagrams

Having entered the de Bruijn submenu by typing d in the main menu, the de Bruijn screen will appear, having the general form of Figure 9. The left hand side of the screen contains an option menu, the right hand side a polygon representing the full de Bruijn diagram. Subdiagrams are shown by altering the colors of the lines. Unfortunately, the diagrams are usually too congested to be of any use.

`1,2,...` - generates a de Bruijn diagram of n stages
`a,b,c,...` - shows cyclic counterimages
`A,B,C,...` - shows all counterimages
`other letters` - show (p,d) cycles
`+,-` - selects the color palette
`?,/` - clears screen, shows menu
CARRIAGE RETURN - exits

To obtain information from the de Bruijn submenu will probably require the use of pencil and paper, or the use of the screen dump program. Although the program lists all the links in the de Bruijn diagram, the ring diagram is generally too cluttered to use directly from the screen and should be redrawn. Usually the resulting diagram can be further simplified, often it contains many redundant loops. Used casually, it still shows whether there will be many or few periods of a given type.

7.3 Sample periods

given that the circle representing the de Bruijn diagram is too congested to be of practical use, some alternatives are provided which deliver the information in a more usable form. For convenience, they are shown in a small pop-up menu which appears when the de Bruijn calculation has finished; it occupies the region marked "print" in Figure 9.

The options are

`p` - Print a list of all the links in the diagram.
`r` - Show a random walk through the diagram
`s` - Scan all the random walks according to initial link
CARRIAGE RETURN- return to the de Bruijn menu

As the presence of the option CARRIAGE RETURN suggests, one is dealing with a mini-mini menu, which allows some exploration before exiting. Unfortunately, one often forgets the level in which one is operating, and becomes confused.

According to the program and its date of revision, the option `p` will either display the links on the screen, or copy them to a disk file. The screen display halts after each screenfull of information, so that the results can be copied down or saved via a screen dump. It is a terminal option, returning to the de Bruijn menu on completion.

The option `s` may be interrupted at any time, although the presentation of the evolution panel will always be completed. Then, either after finding a satisfactory `r` or having halted at an interesting `s`, two CARRIAGE RETURN's will return one to the main menu while retaining the same line image. By proceeding to the main menu option `g`, the configuration's evolution can be followed indefinitely. Usually it begins to degrade, given the fact that the configuration is rarely compatible with the periodicity of the screen.

7.4 Cycles

Temporal behavior is revealed by the de Bruijn diagrams, but the de Bruijn submenu also contains a provision for calculating spatially periodic configurations. By using the cycle option, `y`, within the limits of memory capacity and reasonable running time, all the periodic configurations of a given width can be enumerated. They are displayed on a separate screen which overwrites the de Bruijn screen.

8 Ancestor menu

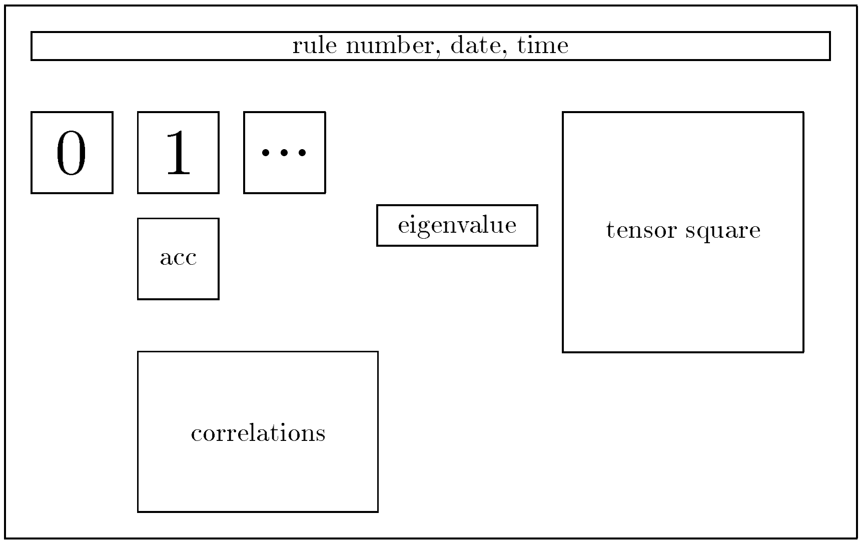

Figure 10: nominal ancestor screen

The number of ancestors of a configuration can be calculated with the help of matrices derived from the de Bruijn diagram. The row and column indices of these matrices are the partial neighborhoods of the automaton; therefore their dimension depends on the number of states and size of the neighborhood. for this reason, an LCAU program only contains the ancestor option when this combination is sufficiently small.

Unless a configuration is infinite, evolution always reduces its size; the missing information is provided by the indices of the de Bruijn matrix. Thus the row index describes the slightly larger left margin of the ancestral configuration, the column index describes the right margin. Periodic configurations depend only on the diagonal elements, those with quiescent support depend only on the (q, q) element, q being the quiescent state.

8.1 General menu

Each state has its own ancestor matrix; products of several ancestor matrices describe the ancestors of sequences of cells. Consequently, the ancestor menu makes provision for a matrix accumulator which holds the matrix counting the ancestors of all the cells which have been entered, running from left to right. Thus, for a binary automaton, `a` places the ancestor matrix for state 0 in the accumulator; `b` does the same for state 1. If there are more states, additional letters are used to generate their matrices.

Corresponding capital letters multiply the accumulator by the appropriate matrix. Thus `aAABBA` would leave a matrix in the accumulator telling how many ancestors the sequence `000110` had, classified according to the states in the margins. Summing *all* the elements of the matrix gives the total number of ancestors, the *trace* tells the number of ancestors in a ring of the given circumference, while the $(0, 0)$ element tells how many are embedded in a field of zeroes.

A very limited amount of matrix arithmetic is provided; for example `=` saves the accumulator and `x` recovers it. The operator `X` multiplies the accumulator by the saved matrix; `Q` squares the accumulator. The latter is useful for obtaining high powers quickly.

The second moment of the ancestor distribution yields the distribution's variance, so the option `t` is included to calculate the sum of the tensor squares of the individual de Bruijn matrices. Since its high powers and their traces are of interest (the power corresponds to the length of the chain whose ancestor is being sought), the operator `T` squares the matrix which it encounters in the tensor square position (the first matrix of the sequence must be introduced by the operator `t`).

Ultimately the largest eigenvalue dominates the powers of all these matrices, which are positive matrices to which the analysis of Frobenius and Perron applies. The options `l` (de Bruijn matrix in the accumulator) and `L` (tensor square) estimate these eigenvalues through successive squaring, reporting the result in a little panel of their own.

`a` - display ancestor matrix for state 0
`b` - display ancestor matrix for state 1
`h` - generate histogram
`j` - classify ancestors by generation
`l` - estimate largest eigenvalue of matrix
`r` - list reversible rules
`s` - summarize the statistics gathered by `S`
`t` - generate and display second moment matrix
`u` - calculate variance of eigenvector components
`v` - calculate variance of column sums
`w` - print extremals from `S`
`x` - recover saved matrix
`A` - multiply by matrix `a`
`B` - multiply by matrix `b`
`Q` - square the matrix

`S` - Survey all rules
`T` - Multiply by tensor square
`X` - multiply by saved matrix
`=` - save matrix in accumulator
`/` - clear the screen
`.` - refresh panel
`?` - display help panel
CARRIAGE RETURN- return to main menu

8.2 Survey options

There are two options which result in fairly extensive computations which are presented on their own individual screens.

8.2.1 Variance histogram

The first of these, corresponding to the option `S`, surveys all of the automata of the class corresponding to the combination (k, r) of the program, with the intention of displaying their variances. Various details emerge from such a calculation; for example all the rules with zero variance are candidates for reversible rules, since zero variance implies that every configuration has the same number of ancestors. As it happens, this is a necessary but not a sufficient condition for reversibility.

Sometimes there is an evident gap between rules of zero variance and the others; unfortunately the gap narrows as the complexity of the automaton increases, so there is no general "gap theorem." In any event, the overall shape of the histogram is interesting.

The ancestor matrices do not directly determine the first moment of a distribution, nor do the sums of their tensor squares determine the second moment. In both cases, the moments arise from taking the traces of powers of the respective matrices, so the quantity of principal interest is the largest eigenvalue, which will eventually dominate all large powers.

8.2.2 Classification by length

The second survey, governed by the option `j`, lists the number of ancestors in more detail, according to the length of the configuration. The ancestor matrices for the specific rule chosen in the main menu are constructed, following which the number of ancestors for configurations of increasing length are calculated. Due to the increasing number of configurations involved, only the very shortest intervals can be presented.

8.2.3 Enumeration of reversible rules

for the smaller combinations of k and r it is possible to make an exhaustive survey, to find which rules are reversible, and to ascertain the reversing rule. Given the fact that the neighborhood length of a pair of inverses does not have to be the same, the surveys performed in the ancestor menu will not necessarily be complete.

9 Undocumented commands

Additional commands are present in different programs, but they are not publicized because they are generally experimental. In future versions of the programs they may be officially documented. Anyone persisting in a desire to use them at their own risk may discover them by studying the source code. The following list is fairly reliable, although the options do not appear on any of the screens presented to the user:

9.1 Field preparation and analysis

`.` - the line evolves for one generation
`=` - repeat the first 40 cells 8 times
  - repeat the first 20 cells 16 times
`D` - display all stored rules in sequence
`Y` - symmetrize the rule
`Z` - (sometimes) clear the line

The instruction `.` is particularly useful when an interesting structure has been found on the general screen, but it is evolving too fast or it is hard to pick out fine details clearly at the resolution of the screen. By returning to the main menu (using any keystroke followed by `y`) the evolution can be followed one generation at a time, and then recorded by a screen dump or noted by hand on a scrap of paper.

Similarly it is possible to transfer the results of the de Bruijn diagram or any other interesting string to the line menu, then return to the main menu to follow their evolution closely with the dot command.

The instruction which repeats the initial segment of the line can be used to increase the variance in some of the probabilistic studies, and to force the automaton to enter a cycle sooner than otherwise. Nevertheless, for most rules a ring of 20 cells is still much too long to reach a cycle within a thousand generations.

9.2 Generating a demonstration

In contrast to the old programs, the new programs contain very few sample rules. By recompiling, one could add further examples that seemed worth preserving in any event the command `D` allows the samples to be reviewed and thus is useful for presenting demonstrations. No doubt some future edition of these programs will allow rule files to be read from disk, as well as allowing the storage of interesting rules which have been found during the course of execution of a program.

9.3 Selecting a video display mode

To ensure a common denominator, the LCAU programs have been written for PC and PC-clone type microcomputers with a CGA video controller. They will work without further alteration on all the compatible video controllers, such as EGA and VGA, of course with neither increased resolution nor additional colors.

A complication arises while working with cellular automata of half-integral radius, which means to say, those whose neighborhoods are of even length. To align the new generation with the old, it should be shifted by half a cell width. With a given (low) resolution there is no alternative to shifting the line, but with higher resolution, two consecutive pixels can be alloted to each cell. Shifting is accomplished by changing the parity of the assignment.

The representation of automata with five or more states likewise presents a problem, given that CGA can only show four colors, including black. Using larger pixels with composite colors reduces the number of cells in a video display which does not bear close scrutiny.

A video controller which goes under the trade name of PLANTRONICS offers doubled resolution with sixteen colors, amongst its other options. In most of the LCAU series, provision has been made for the PLANTRONICS mode. Options in the main menu provide for switching modes.

! - enter PLANTRONICS mode
* - return to CGA mode

For binary automata, where additional colors are not needed, and where monochromatic pixels can still be split by going to higher resolution, a different selection of options exists, yielding different color schemes.

* - white cells on red background
+ - white cells on black background
- - blue cells on black background (normal)

As yet, no reconciliation has been made between the PLANTRONICS mode and either EGA or VGA. PLANTRONICS mode is implemented by setting a flag at the same time that the mode assignment is loaded into the output port of the controller, which is the only element of hardware involved. The TURBO C functions PEEK and POKE are used to deposit information in the video memory area when the PLANTRONICS mode is active.

9.4 Product rules

Unless the number of states of an automaton is a prime number, the possibility exists of defining a product automaton. Although such automata occur naturally amongst all the rules of a given type, it is convenient to be able to generate them directly from the rules defining their factors.

P a b - direct product of Wolfram rules a and b

9.5 Reversible rules

Edward Fredkin has proposed a type of direct product automaton which is reversible. Provision exists in the appropriate LCAU programs for generating the rule table for such automata, as well as the tables for their inverses.

m a - Fredkin automaton using rule a and XOR
M a - Inverse automaton using rule a and XOR
n a - Fredkin automaton using rule a and XNOR
N a - Inverse automaton using rule a and XNOR

9.6 special commands for rule design

Some special commands, are likely to be discarded in future versions of the programs in which they appear.

`U` - push rule and flags
`V` - pop rule and flags
`G` - fetch rule and flags without popping
`x` - random values for unflagged transitions
`X` - random values for all transitions
`1` - some particular transitions
`2` - other particular transitions

9.7 options within rule minimenus

Besides cursor placement and state definition, the minimenus involved with rule definitions sometimes have additional options. When the rule is very long, tabs and margin seeking options are convenient.

0,1,2,3, ... - define transition
tab - next quad = cursor fast advance
space - advance cursor
backspace - retract cursor

In situations where incremental definition is convenient, markers are useful to distinguish between transitions that have been fixed and those which are still under consideration.

up arrow (MOU) - set flag
down arrow (MOD) - remove flag

Another, although related, use for markers is to protect the marked states from the random number generator, so that the consequences of varying only a portion of a rule can be studied.

10 REC programs

First, REC stands for *regular expression compiler*, a rather fanciful term created one day when the program had to be given a name quickly. Anyway the idea was to make programs follow the style of regular expressions, using elements of concatenation, selection of alternatives, and iteration. It was created for the purpose of completing an acceptance test for a PDP-8 (and consequently had to be very compact) on the basis of earlier work with "operator predicates" in LISP.

10.1 The language

The essence of REC is the use of four symbols of control (the two parentheses, colon, and semicolon) together with operators and predicates. Operators are just subroutines; predicates have a truth value in addition.

As usual, a pair of balanced parentheses is used for delimiting groups of symbols, but the group is also to be regarded as a predicate whose truth value depends on how its execution was terminated. Execution proceeds from left to right in the normal manner of reading English text, although the sequence can be interrupted by predicates, colons, and semicolons; all REC programs must be parenthesized.

A colon signifies repetition from the opening left parenthesis of any given level, being the embodiment of iteration in REC . On the other hand a semicolon implies proceeding at once to the closing right parenthesis, while assigning the value "true" to the process just completed. In point of fact one does not include the right parenthesis as part of the departure, but continues from the symbol just beyond it; meeting a right parenthesis as a normal part of a sequence also terminates the subexpression, but assigns it the value "false."

This leaves the role of predicates in a REC expression to be explained. Regular expressions themselves are indeterminate, since they describe the whole class of strings obtained through arbitrary choices of alternatives and iterates; a computer program requires a concrete choice at each juncture. Predicates provide the decision; when true the sequence of symbols continues without interruption, but falsity implies skipping to a new program segment.

The new segment is the one which immediately follows the nearest colon, semicolon or right parenthesis at the same parenthesis level. In the case of the parenthesis, the value "true" is assigned; this is the mechanism through the boolean complement of a truth value may be achieved. In other words, if p is a predicate, (p) is its logical negative. Likewise, when p and q are predicates, (p;q;) corresponds to p or q and (pq;) to p and q.

Arbitrary boolean combinations are possible; for example exclusive or corresponds to (p(q);q;), but one must beware that q could get executed twice without being reproducible (for example, by reading the keyboard twice). Such dilemmas are not frequent, but they do occur and lead to schemes for preserving information, defining variables, and what not; none of these are dealt with at the level of skeletal REC programming.

10.2 Defining programs

Within the basic skeleton the programmer must provide for the individual operators and predicates, generally through a symbol table indexed by ASCII characters. REC is not required to work with single character symbols, but the complexity of parsing a program in the compiler is increased by a whole order of magnitude if anything but the simplest variants on this restriction are allowed.

Over the years extensions to REC have occurred, and specialized applications have developed. The most important extension provides for creating new subroutines. They are designated by a composite symbol such as @s; "at sign" joins the list of reserved letters. Subroutines are defined by enclosing a list of definitions (followed by a main program) within braces:

(...) s (... s ...);

in fact such a braced list may be included anywhere in a REC expression that operators, predicates, or subexpressions can occur. The scope of the definitions lies exclusively within the braces; the same symbols can be reused elsewhere or even recursively.

Certain additional features have proved convenient, but are common to most programming languages. One is to ignore space, tabs, and so on unless they are an actual part of quoted data; the other is to permit comments. In REC , comments are enclosed recursively within square brackets, which permits disabling debugging or trial segments without removing them completely.

It is hard to work with programs which are incapable of providing data to themselves; some form of quotation usually has to be provided. In some contexts, recognizing numbers as such and transforming them into a binary representation is essential; in others quoted character strings suffice.

A counter, written !n!, is a useful predicate, true n times and then false; (!12!x:;) would perform the operation x a dozen times.

The two most important levels of specialization that we have worked with include, first, introducing a pushdown list and, second, introducing a workspace.

The whole mechanism of numerical processing goes well with the first; one predefines arithmetic operations, comparisons, input and output conversion, and so on, as appropriate operators and predicates. Fixed subroutines attend to such things as square roots or trigonometric functions, just as is done in other programming languages. When specific calculations are required, they are performed by writing programs.

Dealing with arbitrary arrays requires further machinery; consequently an arithmetic REC has a closer affinity to APL than to FORTRAN. Nevertheless fixed arrays are convenient, from which a matrix version of REC can be devised.

The second extension really produces an exotic structure, suitable for editors, compilers, and the like. However, both of these versions of REC are much more ambitious than what is included in LCAU.

10.3 Operators and predicates

Part of the need for something like REC comes from the proliferation of options in a program, and in finding some way to deal with them. As written, most programs already contain options to perform the simple and basic actions. To combine them requires careful attention from the keyboard; so REC serves pretty much in the role of a universal macro implementation. It contains all the control structure which is required, wherein the previously existing options can serve as the operators. Predicates must be introduced; these can take the form of counters, responses to pressing a key, or some internal condition within the program such as a vanishing field.

10.4 The REC menu for PROBKR

The menu of REC operators, predicates, and control symbols is defined within the header file RECTBL.H; it is similar to the keyboard menu except for being more concise, and including the REC control characters.

```
    " space                         "
    "! - counter in form !n!        "
```

```
"@ - Subroutine                    "
"# - Integer parameter             "
"$ - Double parameter              "
"% - 10%  T predicate              "
"& - 50%  T predicate              "
"'                                 "
"( start of expression             "
") end of expression               "
": - repeat the subexpression      "
"; - exit subexprn with value T    "
"@x - call the subroutine x        "
"A - strip of 3 plane sections     "
"B - define cell density (mils)    "
"E - show one plane section        "
"G - point of return map in color  "
"K - contours of rand-totl match   "
"O - select dimension of automaton"
"Q - generate cubical checkerboard"
"R - empirical mean field w/delay  "
"T - #n# T for totalistic rule n   "
"[ - start of comment              "
"] - end of comment                "
"_ - panic exit                    "
"a - a-priori probabilities        "
"b - create random line (pl mils)  "
"d - insert diagonal in graph      "
"f - place frame around graph      "
"g - one point of return map       "
"i - iterate 2d gen mean f. curve  "
"k - keyboard interrupt request    "
"n - totalistic sum frequencies    ",
"o - degree of randomness (nbrhd)  "
"p - show cell density in panel B  "
"t - contour of s.c. 2-block freq  "
"x - show nth gen mean field curve"
"z - clear one of the panels       "
"{ - start of program              "
"} - end of program                "
"~ - show parameter panel          "
```

11 Design of rules

To get an idea of how to use some of the features of the LCAU programs, consider the exercise of designing a binary counter for a (4,1) automaton. The reason for choosing a quaternary automaton is that the two states of the counter, “0” and “1”, can occupy two states of the automaton, the quiescent background another, with one more state left over to represent a carry; this state can be mobile—a glider.

This automaton can be devised using the line editing portion of the main menu, in interaction with the rule display occupying its niche in the same menu.

11.1 The counter itself

Let us first discuss the design of the counter.

11.1.1 Infrastructure

To begin with, a quiescent background must be established, against which ZEROES, ONES, and CARRIES can be distinguished.

quiescent background

$$\begin{array}{cccccccccl}
. & . & . & . & . & . & . & . & . & \\
. & . & . & . & . & . & . & . & . & \varphi(0,0,0)=0
\end{array}$$

first bistable state - ZERO

$$\begin{array}{cccccccccl}
. & . & . & 1 & . & . & . & . & . & \varphi(1,0,0)=0 \\
. & . & . & 1 & . & . & . & . & . & \varphi(0,1,0)=1 \\
. & . & . & 1 & . & . & . & . & . & \varphi(0,0,1)=0
\end{array}$$

left running glider - CARRY

$$\begin{array}{cccccccccl}
. & . & . & . & 2 & . & . & . & . & \varphi(2,0,0)=0 \\
. & . & . & 2 & . & . & . & . & . & \varphi(0,2,0)=0 \\
. & . & 2 & . & . & . & . & . & . & \varphi(0,0,2)=2
\end{array}$$

second bistable state - ONE

$$\begin{array}{cccccccccl}
. & . & . & 3 & . & . & . & . & . & \varphi(3,0,0)=0 \\
. & . & . & 3 & . & . & . & . & . & \varphi(0,3,0)=3 \\
. & . & . & 3 & . & . & . & . & . & \varphi(0,0,3)=0
\end{array}$$

Up to this point, ten transitions have ensured that desired movement or lack of movement thereof takes place; also that spurious growth or expansion does not take place around isolated cells.

11.1.2 Binary arithmetic

The next group of definitions shows the glider interacting with the bistable cell as a carry would propagate through a counter. In half of the cycle, ZERO absorbs a carry, changing to a ONE. In the second half ONE reverts to ZERO, generating a carry.

flip ZERO, absorb CARRY

$$\begin{array}{ccccccccc l}
. & . & . & 1 & . & . & 2 & . & . & \\
. & . & . & 1 & . & 2 & . & . & . & \varphi(1,0,2)=2 \\
. & . & . & 1 & 2 & . & . & . & . & \varphi(0,1,2)=3 \\
. & . & . & 3 & . & . & . & . & . & \varphi(1,2,0)=0 \\
. & . & . & 3 & . & . & . & . & . &
\end{array}$$

flip ONE, propagate CARRY

$$\begin{array}{ccccccccc l}
. & . & . & 3 & . & . & 2 & . & . & \varphi(3,0,2)=2 \\
. & . & . & 3 & . & 2 & . & . & . & \varphi(0,3,2)=2 \\
. & . & . & 3 & 2 & . & . & . & . & \varphi(3,2,0)=3 \\
. & . & . & 2 & 3 & . & . & . & . & \varphi(0,2,1)=0 \\
. & . & 2 & 1 & . & . & . & . & . & \varphi(2,0,1)=0 \\
. & 2 & . & 1 & . & . & . & . & . & \varphi(0,2,3)=1 \\
2 & . & . & 1 & . & . & . & . & . & \varphi(2,3,0)=0 \\
. & . & . & 1 & . & . & . & . & . & \varphi(2,1,0)=1
\end{array}$$

Eleven more transitions have been fixed, but it is still not known how close together the bistable cells can be placed; the critical distance is separation by a single cell, which should certainly remain quiescent. Consecutive placement requires an altogether different design.

11.1.3 Packing bistable cells

If the gap between two ONEs looks no different to a traversing glider than its approach to an isolated ONE, a carry will propagate as far as necessary without any special treatment.

Gap between like cells

$$\begin{array}{ccccccccc c}
. & . & . & 3 & . & 3 & . & . & 2 & . \\
. & . & . & 3 & . & 3 & . & 2 & . & \varphi(3,0,3)=0 \\
. & . & . & 3 & . & 3 & 2 & . & . & \varphi(3,2,1)=3 \\
. & . & . & 3 & . & 2 & 3 & . & . & \varphi(2,3,1)=0 \\
. & . & . & 3 & 2 & 1 & . & . & . & \varphi(3,1,0)=1 \\
. & . & . & 2 & 3 & 1 & . & . & . & \varphi(1,0,1)=0 \\
. & . & 2 & 1 & . & 1 & . & . & . & . \\
. & 2 & . & 1 & . & 1 & . & . & . & . \\
2 & . & . & 1 & . & 1 & . & . & . & .
\end{array}$$

Gap between unlike cells

$$\begin{array}{ccccccccc c}
. & . & . & 1 & . & 3 & . & . & 2 & . \\
. & . & . & 1 & . & 3 & . & 2 & . & \varphi(1,0,3)=0 \\
. & . & . & 1 & . & 3 & 2 & . & . & \varphi(1,2,1)=0 \\
. & . & . & 1 & . & 2 & 3 & . & . & \varphi(3,0,1)=0 \\
. & . & . & 1 & 2 & 1 & . & . & . & \varphi(0,3,3)=2 \\
. & . & . & 3 & . & 1 & . & . & . & . \\
. & . & . & 3 & . & 1 & . & . & . & .
\end{array}$$

In total, twenty nine transitions suffice to simulate a binary counter, but there is still no source for the carries which operate it. Isolated bistable cells form its body; packed at their densest, quiescent cells must still separate them. Four generations are required to pass on a carry; therefore one glider every fourth cell is the closest feasible packing for gliders.

Thirty five transitions were not used in defining the basic counter, leaving them free to form an operating environment. Isolated cells and quiescent regions having been preempted by the counter, the remaining transitions necessarily describe the evolution of clusters.

11.1.4 Carry generator

A glider gun will exercise a counter which accompanies it; to obtain a stream of gliders requires exposing the glider cell at the left margin of its cluster from time to time. Some trial and error yields a period four gun constructed from a pair of cells, but there must surely be other viable combinations, as well as guns with longer periods.

a glider gun

$$\begin{array}{ccccccccccc}
. & . & . & . & 2 & 2 & . & . & . & & \varphi(0,2,2)=1 \\
. & . & . & 2 & 1 & 1 & . & . & . & & \varphi(2,2,0)=1 \\
. & . & 2 & . & 1 & 3 & . & . & . & & \varphi(2,1,1)=1 \\
. & 2 & . & . & 3 & 3 & . & . & . & & \varphi(1,1,0)=3 \\
2 & . & . & . & 2 & 2 & . & . & . & & \varphi(0,1,3)=3 \\
. & . & . & 2 & 1 & 1 & . & . & . & & \varphi(1,3,0)=3 \\
. & . & 2 & . & 1 & 3 & . & . & . & & \varphi(3,3,0)=2 \\
. & 2 & . & . & 3 & 3 & . & . & . & & . \\
2 & . & . & . & 2 & 2 & . & . & . & & .
\end{array}$$

11.1.5 Protecting the components

The glider requires only eight additional transitions in its definition, and tolerates crowding by the bistable cells. Nevertheless, the bare gun is susceptible to interference from gliders striking it from the right. Wraparound resulting in the evolution of a cyclic ring will practically guarantee that the glider gun will shoot itself down. Moreover, if there are several glider guns, each will surely attack its neighbor. Defenses can be constructed with the help of transitions as yet undefined.

protection of phase 22

$$\begin{array}{ccccccccccc}
. & . & 2 & . & 1 & 3 & . & . & 2 & & . \\
. & 2 & . & . & 3 & 3 & . & 2 & . & & . \\
2 & . & . & . & 2 & 2 & 2 & . & . & & \varphi(2,2,2)=1 \\
. & . & . & 2 & 1 & 1 & 1 & . & . & & \varphi(1,1,1)=2 \\
. & . & 2 & . & 1 & 2 & 3 & . & . & & \varphi(1,2,3)=3 \\
. & 2 & . & . & 3 & 3 & . & . & . & & .
\end{array}$$

protection of phase 13

$$\begin{array}{ccccccccccc}
. & . & . & . & 2 & 2 & . & . & 2 & & . \\
. & . & . & 2 & 1 & 1 & . & 2 & . & & . \\
. & . & 2 & . & 1 & 3 & 2 & . & . & & \varphi(1,3,2)=3 \\
. & 2 & . & . & 3 & 3 & 3 & . & . & & \varphi(3,3,3)=2 \\
2 & 2 & . & . & 2 & 2 & 2 & . & . & & .
\end{array}$$

protection of phase 11

$$\begin{array}{ccccccccccc}
. & 2 & . & . & 3 & 3 & . & . & 2 & & . \\
2 & . & . & . & 2 & 2 & . & 2 & . & & . \\
. & . & . & 2 & 1 & 1 & 2 & . & . & & \varphi(2,0,2)=2 \\
. & . & 2 & . & 1 & 3 & 3 & . & . & & \varphi(1,1,2)=3 \\
. & 2 & . & . & 3 & 3 & . & . & . & & .
\end{array}$$

These final adjustments leave a self-consistent mixture of counter arrays and glider guns, inasmuch as the guns are protected from the gliders which they generate, or which can reach them across a field of bistable cells. Eighteen transitions remain to be assigned, which can be used for complications not yet foreseen, such as closely spaced glider guns or a barrage of gliders denser than can be fired from the guns. A complete solution to this problem in neither necessarily possible not unique should it exist; searching for one would be an interesting exercise.

some additional protection

$$\begin{array}{ccc}
\varphi(2,3,2)=1 & \varphi(3,3,1)=2 & \varphi(3,1,3)=3 \\
\varphi(1,1,3)=2 & \varphi(0,1,1)=1 & \varphi(2,0,3)=0 \\
\varphi(1,2,2)=3 & \varphi(3,1,1)=2 & \varphi(2,3,3)=2 \\
\varphi(2,2,3)=1 & \varphi(3,2,2)=0 & \varphi(3,3,2)=2
\end{array}$$

Specific details of these last transitions have not been shown because they are not critical to the operation of the counter, nor they as well defined nor unambiguous as the other transitions. Experimentation with different packings of glider guns or different responses to glider barrages may well be repaid with a more interesting design than the one shown.

11.2 Table of transitions

Any design can be accomplished by hand using paper and pencil, a procedure quite prone to mistakes. Options which have been added to both the line editor and the rule editor in LCAU41 will render mechanical assistance. Transitions can be marked, evolution can be evaluated generation by generation, and account can be kept of the transitions which have already been defined relative to those which are still available for assignment.

Fortunately, this example yielded a straightforward design, its principal objectives having been met before any significant choices had to be made. Given the basic style, counting and carrying have unique solutions; the selection of the four pairs comprising the glider gun was somewhat more arbitrary, although we knew that the glider could only be exposed on the left wall once per cycle, and that the other two states form stable walls.

Still, were the design to be pursued to its ultimate details, most of the work would be done in harmonizing combinations which would occur very rarely, supposing that the original intentions of the design were to be respected in choosing applications. Finishing of these details could require a tremendous amount of backtracking, making mechanical assistance imperative, and requiring its eventual incorporation in the design process.

In any event, the following table summarizes all the transitions. Those defined at stage 14, or accompanied by a dot indicating that they are still unresolved, are subject to revision by anyone seeking the ultimate in perfection. Each neighborhood is accompanied by its image and, within parentheses, the stage at which it first appeared.

000	0	(1)	100	0	(2)	200	0	(3)	300	0	(4)
001	0	(2)	101	0	(7)	201	0	(6)	301	0	(8)
002	2	(3)	102	2	(5)	202	2	(13)	302	2	(6)
003	0	(4)	103	0	(8)	203	0	(14)	303	0	(7)
010	1	(2)	110	3	(9)	210	1	(6)	310	1	(7)
011	1	(14)	111	2	(10)	211	1	(9)	311	2	(14)
012	3	(5)	112	3	(13)	212	.	.	312	.	.
013	3	(9)	113	2	(14)	213	.	.	313	3	(14)
020	0	(3)	120	0	(5)	220	1	(9)	320	3	(6)
021	0	(6)	121	0	(8)	221	.	.	321	3	(7)
022	1	(9)	122	3	(14)	222	1	(10)	322	0	(14)
023	1	(6)	123	3	(10)	223	1	(12)	323	.	.
030	3	(4)	130	3	(9)	230	0	(6)	330	2	(9)
031	.	.	131	.	.	231	0	(7)	331	2	(14)
032	2	(6)	132	3	(11)	232	1	(14)	332	2	(12)
033	2	(9)	133	.	.	233	2	(14)	333	2	(11)

Table 2: Transition table for a binary counter.

Realizing that both the body of the counter and the glider gun can be sketched in crude form by relatively few transitions gives some idea of how common these structures must be among automata in general. The leading edge of whatsoever glider determines its principal characteristics—for example, its velocity of propagation—just as the boundaries between a still life and a quiescent region break the field up into stable domains. Relatively few transitions establish the simplest gliders or boundaries, making them proportionately common phenomena. Consequently they should be among the first characteristics to be checked out when examining and creating a new rule.

11.3 Using the line editor

Having seen how to design a binary counter using pencil and paper, consider how the same task may be performed using the line editor. In editing mode, the line of 320 cells is regarded as 8 lines of 40 cells each. With respect to evolution, the left and right edges are connected cyclically. The top and bottom margins are not connected; rather, the display is scrolled when necessary.

The line panel has a cursor, which serves as a point of reference. Its movements can be driven by an appropriately programmed mouse, by the keyboard arrows, and to a certain extent by space and backspace. The options can be grouped according to function.

11.3.1 Position cursor

The most general options, which are present in each LCAU program, simply position the cursor.

MOD- move cursor downward
MOU- move cursor upward
MOR- move cursor forward
MOL- move cursor backward
`<` - move cursor to left margin
`>` - move cursor to right margin

11.3.2 Enter state

Every program refers to a certain number of states, which are numbered consecutively starting with zero. Very few programs have more than ten states, so the decimal digits can be reserved; typing one inserts the state and advances the cursor. States inappropriate to a given automaton are simply ignored.

`0,1,2,3, ...` - enter state

Between ten and sixteen states, hexadecimal digits can be used, whose letters usurp any other use of the same symbol.

11.3.3 Maintainance of the line panel

Clearing lines and copying lines are frequently required.

`z` - clear the row containing the cursor
`Z` - clear the whole line panel
`q` - clean up the area (remove colors)
ˆ - repeat the line just above the cursor's line

Additionally, the marking can become so cluttered that it is convenient to restore the symbols in the panel to a neutral color, while still preserving the original symbols.

11.3.4 Defining transitions

Several of the options interact with the rule menu, since one of the important applications of the line editor is to define transitions. The basic operation is the insertion of a definition, but marking it for future reference is an important secondary operation. Marking may distinguish between transitions already settled upon and those yet to be defined; it also serves to protect the designated transitions from the random number generator.

`=` - insert transition defined by cell under cursor
`*` - insert transitions for the whole line at once

`x` - unmark all transitions

`+` - set marker for transition under cursor

`-` - remove marker for transition under cursor

In referring to the "transition under the cursor," a neighborhood of the appropriate length is taken from the line just above the cursor, cyclically wrapped if the cursor is near the margin. To make sense, the cursor must be in the second row or lower.

11.3.5 Trial evolution

In order to judge the effects of the definitions, and to verify that the evolution is following the intended course, several degrees of trial evolution are offered as options.

`.` - point evolution (only the cell under the cursor)

`?` - evolution of the whole line

`/` - conditional evolution

The choice of the two symbols `?` and `/` was not particularly elegant—they sit at the lower right hand corner of the keyboard, where they are readily accessible.

Point evolution consults the rule table to find out what the cell under the cursor will become, if no changes are contemplated. It is useful for checking that a planned transition is consistent with those already defined.

Unconditional evolution, via `?`, is the variant which processes the whole line at once, rather than individual cells. The table is taken as given, and followed unquestioningly. Conditional evolution, on the other hand, inserts an "undefined" marker for each unmarked transition in the table. The marker is usually `k` for a k state automaton, for which the only legitimate symbols are `0, 1, ..., k-1`.

By this mechanism, transitions which have yet to be defined stand clearly in evidence. If *any* ancestor of a cell is undefined, the cell will be undefined also; conditional evolution may be safely followed through several lines.

11.3.6 Consistency

Rather than building up an evolution generation by generation, it is possible to check a given field for consistency. This is done by giving distinctive colors to cells derived from their ancestors in the previous line according to the rule table, and those which are in violation of the table.

`c` - test consistency everywhere

`C` - test consistency for marked neighborhoods

Consistency may be judged according to the entire table, or only according to the marked transitions.

11.3.7 Prescribed evolution

The final option concerns specifying two lines in the line editor's array, then searching for a rule which will lead from the first line to the second. Again, certain transitions may be forced by previously inserting them in the rule table and marking them.

`m` - try to match a given evolution

When a rule complying with the requirements is found, the program pauses, awaiting a decision to accept the result, or to continue searching for another, if any. If, at any stage, no rule can be found, the program announces this fact when it pauses.

12 Chaté-Manneville automata

An exception to the general rule that all the LCAU automata are one dimensional has been made for some of the binary automata with larger neighborhoods. The programs are basically one dimensional. On the one hand the neighborhood sizes are inconveniently large, especially for the de Bruijn and ancestor menus; on the other, the volume of the neighborhood coincides with that of various higher dimensional automata whose statistics are interesting.

As a consequence, programs such as LCAU23 and LCAU24 have been set up to concentrate on the probability menu, even though the other menus function partially or not at all. Beginners may simply want to use the REC demonstration, ignoring the rest of the program.

12.1 Using these programs

As presently formulated, there is only one REC demonstration, designed to show off the Chaté-Manneville Swiss cheese, to which the program has been initialized. After loading the program, a Copyright Notice will be displayed long enough for recognition and the first keypress. Normally the delay would be used to randomize the startup demonstration, which makes no difference here.

The main menu shows up in text mode; it may be passed over by typing `t` to get to the probabilistic menu, or one may linger over such things as the REC menu and editor, or the definition of a rule. To get directly to the demonstration, type `t` followed by `f5`. Having tired of the demonstration, it can be halted (say by typing `carriage return`) and other aspects of the program explored. Thereafter use REC demonstration #2 rather than #3 to avoid losing new rule numbers, or use #3 to go back to the original demonstration.

To quit from the probability menu, which incidentally is a graphics screen, type `carriage return` followed by `q`. Otherwise use `q` from the main menu.

Since an investigation of the conflict between totalistic rules and mean field theory is the principal objective of this series of programs, there are many options which vary the dimension of the automaton, as well as the degree of randomness in the connections of a one dimensional automaton. There is just one REC demonstration (number 4) which assesses the relation between neighborhood frequency and cell frequency by jointly plotting the return map and a contour map showing the fit between the two as equilibrium density and percent admixture are varied.

Otherwise, one must have a background in the issues involved, and consult the source listings, to derive further information from the programs. The easiest place to begin is by varying the underlying rule, which can still be done while the program is running.

12.2 Keyboard options

The following are the keyboard options in the PROBKR programs (for $k = 2$ and r near 4); all such programs have a uniform menu at the present time.

The screen is divided into six panels. The topmost identifies the automaton and echoes the keyboard command. Immediately below it is the E panel, which can show lines of evolution of the linear automaton. Immediately below that is the large M panel, also screenwide, used for showing the menu, parameter selection, and varied data. The bottom half of the screen is divided into three panels, A, B, and C, in which various graphs can be shown.

The layout is the same one shown in Figure 6.

Panel A generally holds two-dimensional graphs or histograms of cumulated data; panel B often shows data as a function of the step number, panel C is devoted to the return map and associated relationships.

'+' - select white-cyan-magenta palette.
'-' - select yellow-red-green palette,
'a' - shows a priori probabilities and other items in M panel.
'A' - multidimensional evolution shown via triple sections in panel A.
'b' - generates a random line with pl mils of ones.
'B' - Bn loads parameter pl with n (mils), whose default is 500.
'c' - cn shows 8 iterates of probability n (mils) in panel C.
'C' - Cn shows 100 iterates of probability n (mils) in panel B.
'd' - dn shows 100 iterates after a delay of 2500 in panel B.
'D' - Dn shows 100 iterates after a delay of 50000 in panel B.
'e' - displays 12 generations of evolution in panel E.
'E' - multidimensional evolution shown via plane sections in panel A.
'f' - generates a random rule with pr mils of ones.
'F' - Fn loads parameter pr with n (mils), whose default is 500.
'g' - shows 50 generations of evolution in red in panel C.
'G' - shows 200 generations of evolution in green in panel C.
'.' - shows one generation of evolution is red in panel C.
'h' - graphs entropy versus probability in panel C.
'i' - shows once iterated mean field probability curve in panel C.
'j' - shows twice iterated mean field probability curve in panel C.
'J' - shows nfold iterated mean field probability curve in panel C.
'k' - shows thrice iterated mean field probability curve in panel C.
'l' - shows fourfold iterated mean field probability curve in panel C.
'L' - fit a parabola to 50 points of empirical return map.
'H' - interpolate a parabola to 3 points of empirical return map.
'm' - evaluates frequencies and pair frequencies for one generation.

'M' - frequencies and pair frequencies for several generations.
'n' - calculates neighborhood frequencies, displays in panel M.
'N' - calculates neighborhood frequencies, displays in panel M.
'o' - on defines randomness and connectivity in dimension 1.
'O' - On defines the dimension of the automaton.
'p' - displays the frequency of ones in each generation in panel B.
'P' - displays a histogram of frequencies of ones in panel A.
'r' - displays 1^{st} vs n^{th} generation return map in panel C.
'R' - displays $(n-1)^{st}$ vs n^{th} generation return map in panel C.
's' - similar to g but plots every third point.
'S' - similar to g but plots every third point.
'X' - set Bernstein modifier bx[bi] (in mils).
'I' - sets the index (i.e., bi) of the Bernstein Polynomial.
'x' - graphs the mean field theory curve in panel C.
'y' - shows second generation mean field probabilities in panel C.
'w' - shows third generation mean field probabilities in panel C.
't' - shows contours of two block selfconsistency in panel A.
'v' - generate some regular points in the cell array.
'V' - generate some other regular points in the cell array.
'1' - histogram of iterated mean field probabilities in panel M.
'2' - histogram of iterated two-block frequencies in panel M.
'3' - histogram of iterated three-block frequencies in panel M.
'4' - histogram of iterated four-block frequencies in panel M.
'5' - histogram of iterated five-block frequencies in panel M.
'6' - histogram of iterated six-block frequencies in panel M.
'7' - histogram of iterated seven-block frequencies in panel M.
'?' - display the option menu in panel M.
'!' - display the parameter menu in panel M.
'z' - zn clears panel n.
'/' - clears the whole screen.
'*' - prints the Bernstein coefficients in panel M.
'#' - # a b c f(a) f(b) f(c) interpolates a parabola to 3 points.
'&' - &n graphs a combination of uniform and mean field curves.
'$' - repeats &n at intervals of 0.1 from 0.0 t0 1.0.
'%' - print differences in totalistic neighborhood frequencies.
'K' - shows a contour plot of neighborhood conformance to & curve.
'@' - show mean field curve on Hiplot plotter.
'^' - initialize the Hiplot plotter.
'Z' - dismiss the Hiplot plotter.
MOD - moves the bernstein pointer down along right margin panel C.
MOU - moves the bernstein pointer up along right margin panel C.
MOR - increases corresponding Bernstein coefficient.
MOL - decreases corresponding Bernstein coefficient.

MOA - reduces Bernstein increment.
MOB - increases Bernstein increment.
MOX - resets all Bernstein parameters.

MOU and so on refer to mouse buttons, which are equivalent to keyboard arrows, control arrows, and escape.

12.3 Analysis of the options

Since the number of options is quite large, it is worth analyzing them by subsets having similar function.

12.3.1 Palette selection

The CGA video control allows either enabling (`+`) or inhibiting (`-`) the blue electron gun. One choice or the other may be more pleasant for some purposes.

12.3.2 Select dimension and randomness

The options `O` and `o` permit variations within the theme of (k, r).

Degree of randomness Random connections are only available in one dimension. The range of n in `on` is the following:

`0` - completely random neighbors
`1` - usual neighbors
`2` - random pairs
`3` - random triples
`4` - random quadruples
`5` - random eighths
`6` - random sixteenths

Select dimension The range of n in `On` depends on the automaton; most of the reasonable variants have been provided for. It is a question of whether a cube, face centered lattice, or rectangular neighborhood has the right volume.

12.3.3 Help, information, housekeeping

Although they are frequently out of date, some very terse help panels are available. When the panels become too cluttered, they can be cleared.

Option menu Typing `?` will display a very concise list of the options in the M-panel. The list is not intended to be explanatory, merely to serve as a reminder of what is available. It is not always up to date.

Parameter menu An alternative display in the M-panel, listing the values of several parameters, can be invoked by typing `!`. Alongside the values of the parameters are shown the options required to change their values. Normally the value required is a number, which must be terminated in a manner acceptable to the C language (space or carriage return suffice).

Clear panel Typing `zx` will clear panel x; x should be a single letter chosen from the set {`a, b, c, m, e, x`}, the latter designating the whole screen. `/` performs the same function.

12.3.4 Mean field probability

The change of density of live cells (state 1) from generation to generation may be estimated from the laws of probability, assuming that the probability of any cell being live is independent of that of its neighbor.

Rigorous probabilities It requires time, and a certain amount of memory space, to obtain these probabilities from the rule table. For all rules and all dimensions, the single generation curve is easy to compute. For many, but not all, the second generation is easily obtained; the principal limitation is the rapid increase in the number of neighborhoods with dimension.

`x` - graph mean field curve in panel C.
`y` - second generation mean field probability.
`w` - third generation mean field probability.

Monte Carlo probabilities A more efficient way to estimate the mean field curves, although subject to the usual statistical irregularities, is to follow the evolution of sample fields of given probability and record the results.

`r` - `rn` displays 1^{st} vs n^{th} generation return map in panel C.
`R` - `Rn` shows $(n-1)^{st}$ vs n^{th} generation return map in panel C.

12.3.5 Iterated mean field

If the probabilities in adjoining cells were truly independent, the development of probability from one generation to the next would proceed by iteration. In some cases it does, in others it does not. Several degrees of iteration can be generated for purposes of comparison.

Iteration of the full range To the extent that time and space permit, the mean field curve can be iterated one or more times.

`i` - graph once iterated mean field in panel C.
`j` - twice iterated mean field
`k` - thrice iterated mean field
`l` - quadruply iterated mean field
`Jn` - n-fold iterated mean field

Iteration of individual points It is easier, and sometimes more informative, to calculate the iterates of one single point on the mean field curve. The values can be shown from the beginning, or starting after a delay.

- `c` - cn shows 8 iterates of probability n (mils) in panel C.
- `C` - Cn shows 100 iterates of probability n (mils) in panel B.
- `d` - dn shows 100 iterates after a delay of 2500 in panel B.
- `D` - Dn shows 100 iterates after a delay of 50000 in panel B.

12.3.6 Sample evolution

Although it is possible to return to the main menu and then display the evolution of the automaton on the full screen, often a small excerpt gives all the information that one requires while avoiding the inconvenience of menu shifting. Two options offer this advantage.

One dimension The option `e` places about a dozen lines of evolution in the E-panel, at the top of the probability screen. In doing so it treats the automaton as though it were one dimensional, even though that option is not active, by just copying the field array.

Higher dimension The option `E` places a small square of two-dimensional evolution in the A-panel. Altogether, nine squares are shown before the display begins to repeat; if the automaton is not two-dimensional, then the excerpt shown is a two-dimensional slice through the field of the automaton.

Both displays can be interrupted by a keypress; otherwise they continue through one full cycle before stopping.

Triple section The option `A` is similar to `E`, but shows a triple section, which sometimes aids the interpretation of the section.

12.3.7 Frequencies in succeeding generations

In a panel devoted to probability, it is natural to compare the actual proportion of live cells with the theoretical predictions. Two options carry out the evolution without exhibiting the automaton; they simply place a point corresponding to $(p, \varphi(p))$ in the C-panel. Presumably some option such as `x` has already inserted the theoretical curve into the same panel.

One generation To see the effect of one single step, type `.` to show one single red dot in the C-panel.

50 generations The option `g` (chosen for its similarity to the same option in the main menu) places 50 red dots in the C-panel; these are points on the return map of the mean field curve. Usually they will cluster about an average value, but their behavior in the Chaté-Manneville automata is markedly different.

200 generations Option `G` runs through more generations than `g`, currently four times as many. These figures are parameters which can be altered by editing the source code and recompiling the program.

Every third generation, 50 times Only for those LCAU which correspond to the Chat-Manneville automata, the return map can be shown for every third generation. The reason is that the period 3 is a distinguishing feature of these automata. The option `g` creates this map.

Every third generation, 200 times As an option analogous to options `s` and `G`, the return points can be plotted 200 times by `S`.

Time series Option `p` will show a time series in the B-panel rather than the return map in the C-panel; successive generations are given contrasting colors which is useful in observing periodicities.

Histogram The option `P` shows the same information in the form of a histogram in the A-panel. Peaks in the histogram also indicate periodicity.

12.3.8 Synthesizing a return map

Since the mean field curve does not always follow the empirical data, provision has been made to generate an empirical curve, by prescribing the coefficients of a Bernstein polynomial. Cursor arrows or mouse movements are used to select the coefficient or to establish its value.

MOD - moves the bernstein pointer down along right margin of panel C.
MOU - moves the bernstein pointer up along right margin of panel C.
MOR - increases corresponding Bernstein coefficient.
MOL - decreases corresponding Bernstein coefficient.
MOA - reduces Bernstein increment used by MOL, MOR.
MOB - increases Bernstein increment used by MOL, MOR.
MOX - resets all Bernstein parameters.
* - lists the Bernstein coefficients in panel M.

Provision has been made for a modifier, inspired by the idea of a probabilistic automaton, which can multiply each Bernstein monomial. The default values are all 1.0.

`X` - set Bernstein modifier bx[bi] (in mils).
`I` - sets the index (i.e., bi) of the Bernstein monomial.

Another service which has been provided is to fit a parabola to a portion of the return map, the intention being to see how closely the dynamics of the classical logistic function can be matched. These options are very specific to the Chaté-Manneville automata, and must be used cautiously; otherwise floating point overflow is likely to occur. Some control over the points may be obtained by generating an initial field of known probability.

`L` - fit a parabola to 50 points of empirical return map.
`H` - interpolate a parabola to 3 points of empirical return map.
`#` - # a b c f(a) f(b) f(c) interpolates a parabola to 3 points.

The option `#` requires that the coordinates of the points be stated explicitly.

12.3.9 Block probabilities

Exclusively for one dimensional automata, it is possible to develop a set of self-consistent equations expressing the Bayes estimate of the probability of finding sequences of cells, including the mean field probabilities as a special case for single cells. These options are therefore not relevant to the discussion of the Chaté-Manneville automata, but they are present in the probability menus of all the LCAU programs.

Block probabilities The length of block which can be considered is limited by the memory space available, and the required time of computation. The equations are solved by iteration; a histogram of the individual frequencies is shown in the M-panel. To the extent that they are applicable, the options are:

`1` - iterated mean field probabilities
`2` - iterated two-block probabilities
`3` - iterated three-block probabilities
`4` - iterated four-block probabilities
`5` - iterated five-block probabilities
`6` - iterated six-block probabilities
`7` - iterated seven-block probabilities
`t` - contour map of the two-block function

The two-block probabilities are dependent on only two parameters, which can be chosen to be the probability of a live cell and the probability of a pair of live cells. This accounts for the presence of the option `t` in the list, whose contour appears in the A-panel.

Frequencies and pair frequencies The option `m` and its counterpart `M` evaluate frequencies and pair frequencies for respectively one and several generations. The purpose of the latter is to gain better statistics by averaging. The results are reported in the M-panel.

Neighborhood frequencies The options `n` and `N` gather statistics concerning the number of cells per neighborhood. Again, the difference lies in the amount of data gathered, and thus the time required to collect it; `N` averages over several generations whereas `n` proceeded generation by generation. Both present their results in the form of a histogram in the M-panel.

***A priori* probabilities** At a level still simpler than mean field theory, the probability of encountering a cell during evolution can be taken to be proportional to the fraction of transitions by which its value can arise. The required scan of the rule table is performed by the option `a`, which also calculates some relevant entropies.

Entropy A general curve of entropy versus probability can be displayed in the C-panel by using option `h`.

12.3.10 Random rule

In order to generate a random rule without going to the main menu and back, an analog of the main option `X` is provided. Moreover, `X` is set at 50%, whereas an arbitrary probability is often needed.

Place rule in table Option `f` sets up a random rule with `pr` percent of live cells, where `pr` is a parameter whose value can be adjusted.

Define parameter `pr` Option `Fxxx` will load the random rule percentage parameter `pr` with the value `xxx`, which should be an integer in the range 0-1000. The value is expressed in mils, therefore 50% probability would be expressed as `F500`; it is the default probability taken by `pr` if no other definition has been given.

12.3.11 Random line

Random lines with a fixed percentage of live cells are also required; again without going to the main menu to use the option `y`, and with fractions which are not necessarily 50%.

Set up a configuration Option `b` generates a field with `pl` percent live cells; it is generated as a single one dimensional array, although it will be interpreted as having whatever dimension that has been set by the option `O`.

Define parameter `pl` Option `Bxxx` will load the random line percentage parameter `pl` with the value `xxx`, which should be an integer in the range 0-1000. The value is expressed in mils, therefore 50% probability would be expressed as `B500`; it is the default probability taken by `pl` if no other definition has been given.

12.3.12 Hiplot services

To create a permanent record of some of the return maps, subroutines have been added to service a HOUSTON INSTRUMENTS plotter, or any other model which responds to the same commands. There is nothing in the subroutines other than the generation of a stream of ASCII characters directed to a serial output port, but the characters have to be the ones which will activate the plotter.

`^` - initialize the Hiplot plotter.
`@` - `@1` shows, `@1` suppresses mean field curve
`Z` - dismiss the Hiplot plotter.

12.3.13 Miscellaneous

Given that the LCAU programs are used in ongoing research, the list of options will vary from time time; this is particularly true in view of the fact that the alphabet of options is now nearly full.

One item of research has concerned the fact that totalistic rules seem to follow a different probability distribution, distinct from that of general rules. At one point this was considered to be an explanation of the Chaté-Manneville effect, but was later seen to have a secondary importance.

`&` - &n graphs a combination of uniform and mean field curves.
`$` - repeats &n at intervals of 0.1 from 0.0 t0 1.0.
`%` - print differences in totalistic neighborhood frequencies.
`K` - shows a contour plot of neighborhood conformance to & curve.

Another item of research concerns the details of the mean field curves for automata in Wolfram's Class IV, such as *Life*, which can be studied in LCAU24.

Some of the scarce remaining symbols have been allocated to generating specific regular configurations or specific rules (such as *Life*).

`v` - generate some regular points in the cell array.
`V` - generate some other regular points in the cell array.

12.4 Function keys

The function keys `f4` and `f5` execute respectively, the edited REC program or the selected demonstration. `F4` and `F5` show up to 40 symbols from these programs, for purposes of confirmation. Paragraph up and down change the demonstration selection, while ^PPU and ^PPD show the first and last; generally one tries to keep the usage of these keys consistent with their use in an editor. All these displays are made in the E panel.

13 Acknowledgements