10 Experiments

10.1 Modeling Environments

A large part of our research of the LtL family consists of watching "movies" of the rules started from a variety of initial states. To do these experiments we use two platforms -- WinCA, a windows-based CA modeling environment, and the latest *Cellular Automaton Machine*, known as CAM-8. Let us describe these two platforms.

WinCA

We did the majority of our computer simulations using WinCA, which is currently being developed by Bob Fisch and David Griffeath. WinCA has provided a huge improvement over the DOS days when just creating an initial state was an arduous task. Its features that we find so appealing are its user-friendly interface, its ability to keep track of the many variables involved in LtL research, and its ability to enable the user to design a wide-range of initial states. Let us describe these features.

WinCA enables one to define and save a CA *experiment*. An experiment consists of the parameters for the CA rule, the lattice or *system size*, the *boundary conditions*, the initial state, the *step size*, the *final time*, and, if desired, a *final state*. Additionally, one has the option to attach *notes* about the experiment. For simulations we work on subsets of $\mathbb{Z}^2$. That is our system, or universe, is $\mathcal{X} = \Gamma \cap \mathbb{Z}^2$ where $\Gamma \subset \mathbb{Z}^2$ is some finite subset. Easily varying the system size is convenient because understanding the dynamics generated by rules near phase boundaries or for large ranges often requires a large system size, while for other endeavors, such as finding the period of a bug, a small system size suffices. It is convenient to use the appropriate system size since large system sizes require huge amounts of computer memory and hence cause the rules to update very slowly. The boundary conditions we use most often are periodic, meaning that opposite edges of the finite lattice are identified. This turns the universe into a torus and so, on the computer screen, the trajectories of live sites may appear to reach the boundaries of the system but they never exit. For some rules, such as those in the thin bug regime, using periodic boundary conditions is a decent way to simulate the infinite system -- somewhere out there, another bug's trajectory will enter the finite window at which we are looking. Another kind of boundary condition we use is to let all sites on the boundary be in state 0. In this way, it is though the lattice is surrounded by an infinite sea of dead sites. In some cases, such as a bug maker sending off an infinite stream of bugs, it

is a convenient way to simulate the infinite system. In other cases this is not appropriate since live sites at the edges of the system will not necessarily always become dead.

Another crucial part of WinCA's design is that it makes constructing initial states very easy. For example, it enables the user to create such initial states as product measure with any density, circles with varying radii, rectangles with varying side lengths, half planes, or some combination of those. It enables the user to save these states as bitmaps and also to save their evolutions after being run under the CA rule. Additionally, it can import initial states from various paint programs. Not only does this allow the user to design practically any desired initial state, it allows one to import images from such places as the World Wide Web. For example, whenever Paul Callahan posts new Life configurations on his Web site (see [Cal]), we can use the "capture" mode on our favorite paint program, Paint Shop Pro, to import the file to the paint program. Once in the paint program, we format it as desired, either save it to the clipboard or as bitmap, and then use it as an initial state in WinCA.

For more information on WinCA, visit the Primordial Soup Kitchen (see [Gri1]). Most of the graphics exhibited there were generated using WinCA. Additionally, the software is available in the Kitchen's Sink.

CAM-8

The CAM-8 Cellular Automaton Machine comes from the MIT Laboratory for Computer Science, and is mostly the creation of Norman Margolus. CAM-8 is the next generation of interactive desktop parallel processor, offering a unique combination of computing power and visualization capability for the study of cellular automaton dynamics. CAM-8 has greatly benefited our exploration of LtL space. It is a bit more complicated to use than WinCA, since creating initial states and varying parameters is not as easy as clicking a button. However, Bob Fisch wrote a program that converts bitmaps into ".pat" files, which are in CAM-8's language. Thus, we are able to design initial states using WinCA and then import them to CAM-8. This enables us to take advantage of the CAM-8's main attraction – *speed*. Let us describe some of the features that make CAM-8 an excellent CA simulator.

CAM-8 is a modular computing device for extremely fast computation of arbitrarily large cellular automaton (CA) systems. Each CAM-8 module can independently store and update as many as 2^{24} cells of a CA. Each of these cells contains 16 bits of information, permitting 65,536 possible states at each cell. Furthermore, each module can update these cells at the rate of 25 million cell updates per second. A SUN SPARCstation is responsible for driving CAM-8 modules by means of a customized controller card on the SUN's internal bus. The SUN is used to initialize modules with the appropriate starting state and update rule

for a particular CA of interest. After initialization, the host signals CAM-8 to begin computation, at which time the machine performs independently of the host.

Modularity of CAM-8 is realized as follows. As many as eight modules can reside in a single box. The box is an enclosure containing sockets for accepting modules, a frame buffer for video output, circuitry for communication among the modules and with the host, and other necessities (such as a power supply and fan). Within a box, the eight modules operate in parallel. Thus, a full box comprising eight modules can perform 200 million cell-updates per second. CAM-8 architecture permits arbitrarily many boxes to be physically arranged and interconnected into a three-dimensional array, and each box in this arrangement computes in parallel with the others. In this manner, one can compute as large a system as desired; the only limitation is the number of modules available for computation. Furthermore, such a system updates at the rate of $25N$ million cell-updates per second, where N is the number of modules in the arrangement of boxes.

The mechanism used by CAM-8 for updating a CA consists of two stages. In the first stage information is kicked, or shifted, throughout the universe. The kicking stage permits any cell of a module to receive state information from any of the 2^{24} cells within its module, or from any cell in a neighboring module that satisfies the property that in each extensible dimension the distance between the two cells is no greater than the extent of a single module in that dimension. Of course, since CA rules are translation invariant, one can think of the kicking stage as taking individual "bit planes" (also known as layers) of state information and shifting each as much as one module distant in each dimension. Each layer may be shifted independently, so that a cell may receive state information from different locations of a very large neighborhood.

After kicking the layers, the bits from each cell, which contain the gathered state information from each cell's neighborhood, are fed into a lookup table. The lookup table accepts 16 bits of input and generates 16 bits of output to be used as the new state at that cell. Thus, the lookup table represents the actual updating rule for the CA. Its contents are precomputed within the host and loaded into each of the modules before the CAM-8 receives the signal to begin computation. Within a module, each cell's new state is computed serially at a rate of 25 million cell-updates per second. Thus, no parallel processing occurs within a module, although each module computes in parallel with the other modules.

Our CAM-8 frame buffer is connected to a VGA monitor that displays either a 512×512 or $1K \times 1K$ portion of the CA, using 256 colors. This frame buffer may also be connected to a VCR for producing videotapes of CAM-8 display output. Thus, one can conveniently generate and record movies of a CA evolution for sharing with others. CAM-8 also has the capability of producing event counts for gathering aggregate information about a

CA. The events to be counted may be detected using arbitrary Boolean functions of the state bits for a cell. Furthermore, one may divide the CA into subregions of any desired size and report the event counts over each individual subregion. In this way, one may generate quantitative evidence of spatial variations within a CA. The host may take these event counts and generate visuals on the fly to convey the presence of any such variations.

For more information on CAM-8, see [Mar]. For more information on cellular automaton machines, see [TM].

10.2 Two examples of LtL research strategies

The search for bugs

As we explained in chapter 7, it is often the case that no bugs appear when a buggin' rule is run starting from a random initial state. It is thus essential to devise techniques that enable one to discover bugs. One such technique is to run a buggin' rule from a random initial state and capture one of the bugs it generates. The bug is then used as an initial state and the same rule is run. As the rule updates, its parameters are varied and if successful, the geometry of the bug also varies. In some cases, this "sculpts" an appropriately shaped bug for a different rule.

Another strategy that is often fruitful is to create finite deterministic configurations, thought up by studying known bugs. For example, in the thin bug regime, many of the bugs are generated from appropriately sized rectangles. In order to decide what "appropriate" means for a given rule it is usually convenient to sketch possible sizes by hand. Then a paint program, or the WinCA "special initialization" feature, is used to insert rectangles, and the rule is run with that as its initial state. Another strategy is to find bugs in one range, and rescale them up or down to other ranges. If successful, one can compare the objects admitted in a variety of ranges, and generalize the findings.

Local periodicity and other extreme behaviors

For extreme behaviors, it is essential to be very careful not to jump to conclusions. For example, there are rules which seem, from random initial states, to be locally periodic. However, if one inserts a large lattice ball consisting of only one of the two states, that set grows to fill the lattice. For some rules, two large lattice balls, one of each state, may both grow and end in a stand-off with boundaries composed of a pattern created by the two states. For other rules, such a competition between states results in the demise of one of them. Thus, it is essential to test these rules using a variety of initial states, including those that are partly

random and partly consisting of deterministic sets composed of one state or the other. For example, a large lattice ball filled with a cut-out of a time step generated by a rule known to be either aperiodic or periodic can be placed on a background of all zeros. Such an initial state tests the ability of the aperiodic or periodic region to grow. As we discussed in Chapter 8, for rules near the phase transitions in LtL space it is often the case that aperiodic or periodic lattice balls need "help" to grow. Without the help, they either die out, or get stuck inside regions that are convex-confined. Such dynamics are reminiscent of the bootstrap growth of the discrete TGM.