Christopher Ariza
Department of Music
Towson University
8000 York Road, Towson, Maryland 21201 USA
ariza@flexatone.net

Automata Bending: Applications of Dynamic Mutation and Dynamic Rules in Modular One-Dimensional Cellular Automata

A cellular automaton (CA) is often considered a software model of a machine (e.g., a computer) that runs a set of procedures (e.g., software). Such a model is implicit in the sophisticated demonstrations of universal computation that cellular automata can perform, such as John von Neumann's model of a two-dimensional, 29-state CA (1966); Edwin Roger Banks's model of a two-dimensional, binary-state CA (1971); and Matthew Cook's model of a one-dimensional, binary state, rule-110 CA (2004). Others, using CA for a wide range of alternative applications, have often followed this approach, using strict and fixed rule sets to produce systems capable of reliable computations. In artistic applications of CA, however, there is no mandate to use "pure" implementations: the question of whether a CA functions as a proper machine, capable of universal computation or otherwise, has no significance on the extraction of artistically useful value streams.

Bending or even breaking the rules of a CA, introduced here as "automata bending," is analogous to the circuit-bending techniques demonstrated by Qubais Reed Ghazala and others (Ghazala 2004; Collins 2006). As with circuits, applying such techniques to CA permits transforming simple and elementary systems into more-dynamic generators. There can be no objective measure of aesthetic advantage in bending an automaton; such an approach, however, expands the parametric interface of the CA, giving the user more-flexible controls to shape and even direct CA evolution. The parametric interfaces of generative systems must be adapted to the needs of composers; this is an important component of research in computer-aided algorithmic composition (CAAC).

The utility and diversity of CA are frequently overstated. For example, Eleonora Bilotta and Pietro Pantano offer the unqualified claim that CA "display rich and complex patterns, whose organization is completely unpredictable" (2002, p. 156). The majority of CA, in many formats, produce trivial repeated patterns or plain "chaotic" randomness (Wuensche 1999, p. 54); it is the minority of CA that display rich and complex behavior. Furthermore, traditional CA are deterministic: they may be capable of producing randomness (Wolfram 1985), but this randomness is entirely predictable. With the application of automata-bending techniques, however, even repetitive or static CA can be used to generate dynamic data.

In the rare cases that CA do exhibit emergent behavior, this behavior is often realized only in large data spaces, covering expanses of cell state histories easily on the order of tens of thousands, if not significantly more. Except in applications of CA data to audio-rate parameter values, practical usage requires obtaining values from the CA on much smaller numerical scales. Automata-bending techniques, as well as flexible methods of extracting values from the CA, reduce the data space necessary to obtain practical value streams.

This article introduces a language-independent string notation for, and a flexible implementation of, a variable format one-dimensional CA. This CA model permits two types of automata bending: random cell-state mutation, and dynamic, probabilistic rule-sets. Numerous formats of one-dimensional CA (such as standard, totalistic, and continuous), as well as all k and r values (within hardware perfor-

Computer Music Journal, 31:1, pp. 29–49, Spring 2007

mance constraints), are supported. The application of CA values to musical parameters is facilitated by diverse methods of masking cell data and extracting one-dimensional value sequences. Rather than the common approach of hard-wiring CA values to fixed mappings or musical parameters, this model promotes a modular approach, making the CA into a flexible and dynamic one-dimensional value generator. These values can be applied to a wide range of musical parameters, such as event amplitude, panning, rhythm, or pitch; synthesis configurations, envelopes, or wave tables; or direct waveform specification. As a modular component, CA parameter inputs may be controlled by lower-level embedded generators, and CA output values may be used as input for higher-level generators. Ultimately, a modular design promotes using CA-derived values in the same manner as a simple component of a modular synthesizer. While the approach presented here may be implemented in a variety of systems, high-level object interfaces to this model are demonstrated in athenaCL (Ariza 2005a).

This article first defines both the CA and the CA specification string. To show the diversity of possible CA formations, examples of generating CA with this specification string are provided. With this foundation, a review of historical applications of CA to music is given. The automata-bending techniques are then defined and demonstrated, table extraction and mapping techniques are explained, and, finally, high-level musical examples are provided.

The Cellular Automata Specification String

Although the basic mechanics of CA are well explained in numerous texts, both with and without a musical context, a brief explanation will be provided here. A CA is a system that produces sequences of discrete states. These states are encoded in a lattice of cells in one or more dimensions. Each cell can have various values, often in the form of integers or real numbers. A new lattice is generated based on the states of the past lattice. To determine the value of each cell in a new lattice, contiguous sections of cells from the previous lattice (often called a neighborhood) are examined. These previous cells are in the same and surrounding positions of the new cell. For example, given a one-dimensional lattice with sequentially numbered index positions (like an array), a new state for a cell at position 3 may be determined by examining the previous cells at positions 2, 3, and 4. A rule set, or a table of all possible previous cell formations, is consulted to determine the new cell value. Thus, if positions 2, 3, and 4 have values 0, 0, and 1, the new cell value is determined by matching these found values to cell formations in the rule set. A rule set is often referred to as a single "rule" or a "code," even though numerous individual rules, for each possible previous neighborhood formation, must be defined.

The specification of a one-dimensional CA requires numerous parameters. If all parameters are exposed to the user, the number and diversity of parameters can make working with a CA cumbersome. To create a flexible CA implementation that can accommodate a wide range of CA formats and presentations, the approach introduced here collapses these static parameters into a "caSpec," a language-independent CA specification string. The caSpec offers a compact and practical approach to notating CA parameters, and it can be implemented in a variety of languages and systems. Although past research has demonstrated CA in diverse applications, a general input notation has not previously been proposed. This new notation will be used throughout this article, which demonstrates the notation while defining CA parameters.

The caSpec is designed for ease of use. Within a caSpec, a user may provide one or more static parameter values; parameters may be specified in any order, and default values are provided for parameters not supplied. Furthermore, redundant or conflicting parameter values are automatically resolved. Parameters are supplied in a language-independent string notation using brace-delimited key and value pairs: `key{value}`. All keys are single characters; values, depending on the parameter, can be integers, real numbers, or strings. The caSpec only accommodates static parameter values: dynamic parameters must be externally specified.

The specification supports many one-dimensional CA formats. The CA format defines the types of rules applied and the kinds of values

those rules operate upon. Four types are considered here: standard, totalistic, continuous, and float. Standard CA (`f{s}`), where the key "format" is abbreviated `f` and the value "standard" is abbreviated `s`, use discrete cell values and use rules that exactly match previous cell formations; totalistic CA (`f{t}`) use discrete cell values and use rules that match the sum of previous cell formations; continuous CA (`f{c}`) use real-number cell values within the unit interval and use rules that specify values added to the average of previous cell formations; finally, float CA (`f{f}`) use floating-point cell values within the unit interval and, like continuous CA, use rules that specify values added to the average of previous cell formations. The low-level implementation of cell-value data types distinguishes continuous and float CA; the significance of this difference will be demonstrated in this article.

All CA must have either a discrete collection or a continuous range of possible cell values, and they must define the number of previous cells used to determine new values. These CA parameters are conventionally defined with the variables *k* and *r*, respectively. In the case of CA with discrete cell values (`f{s}` and `f{t}`), the variable *k* defines the number of possible values. A `k{2}` (binary) CA has cells with two possible values (conventionally encoded as 0 or 1). In the case of CA with continuous values, the *k* parameter is assigned the value zero (`k{0}`). The number of previous cells taken into account for the generation of new cells (the rule neighborhood) is defined by the variable *r*. In the case of a one-dimensional CA, the number of cells taken into account is equal to $2r + 1$. An `r{1}` CA, for example, takes three previous cells into account; an `r{2}` CA takes five previous cells into account. The specification of *r* typically excludes asymmetrical, even-sized rule neighborhoods. Such neighborhood sizes are explored by Wuensche (1999) and can be accommodated within this notation by accepting half-integer fractional values. Thus, an `r{1.5}` CA takes four previous cells into account; an `r{2.5}` takes six previous cells into account.

In addition to the format, *k*, and *r* values, the size, orientation, and presentation of the CA must be specified. Traditionally, a one-dimensional CA is visually presented in a "space-time" diagram, where the cellular array is presented as a horizontal row of cells. This row, in most implementations, is wrapped from the right edge to the left edge, creating an unbound but finite space. Additional rows, representing subsequent generations after the application of rules, are added beneath this initial row. The resulting data, here referred to as a "table," define cell sites on the *x* axis, and time (or steps of rule application) on the *y* axis. Although tabular presentation is convenient, a cylinder more accurately portrays horizontal row wrapping (Wuensche and Lesser 1992, p. 6). The size of the CA space, or the number of cells in a row, is specified in the caSpec with the key `x`; the number of CA evolutions, or the number of times rules are applied, is specified in the caSpec with the key `y`. In both cases, integer values must be supplied. A caSpec that defines `x{100}y{200}`, for example, will create a space of 100 cells upon which rules are applied 200 times.

Creative applications of CA may not need data from the entire CA table. By masking a sub-table from the complete CA table, diverse presentations of the same CA data are possible. A sub-table is defined in a caSpec with keys for width (`w`), center (`c`), and skip (`s`). The width parameter defines the number of columns extracted from the CA table; if the width is not specified, a default value, equal to the size of the lattice (`x`), is provided. The center parameter defines which column in the CA table (where 0 is the original center) becomes the center of the new sub-table. The skip parameter permits any number of initial row evolutions, or rule processing stages, to be skipped and excluded from the sub-table. For example, a caSpec defining `w{1}c{0}` provides a sub-table consisting only of the middle column of the CA table; `w{3}c{-20}s{10}` provides a sub-table consisting only of rows after the first ten generations, and including only the three columns surrounding the twentieth column to the left of the center column.

Finally, a seed, or the values of the initial lattice, must be specified. The composition of the initial row, in most cases, significantly alters the evolution of a CA. The key `i` is used to define initialization configuration in a caSpec. Values may be strings such as `center` or `random`. Values can also be a se-

quence of integers presented as a single string, such as "010111." If this sequence is shorter than the row size, these values are repeated to fill the initial row.

Dynamic CA parameters are excluded from the caSpec. In the CA model presented here, the mutation probability and the CA rule are dynamic parameters. The application of these parameters will be demonstrated in greater detail in subsequent sections. The mutation probability is expressed as a value within the unit interval. The CA rule is expressed as a real number. In the case of CA with discrete values (`f{s}` and `f{t}`), a rule (or a rule set) can conveniently be encoded as an integer (Wolfram 1983). All rule integers are in a range from one to a maximum rule value. In the case of standard CA (`f{s}`), given k and r, the maximum rule value is $k^{(k^{2r+1})}$.

In the case of totalistic CA (`f{t}`), given k and r (and assuming that cell values are in the range from 0 to $k - 1$), the maximum rule value is $k^{(2r+1)(k-1)}$. With a known maximum rule value, a discrete CA can accept any integer as a rule; a value beyond the range of rules is resolved by the modulus of the maximum rule value. Real-number rule values can also be accepted; floating-point values are interpreted as proportional weightings toward the nearest integer. In the case of continuous CA (`f{c}` and `f{f}`), rules are specified as floating-point values between the unit interval (0–1). As with discrete CA, values beyond the prescribed range are resolved by the modulus of the maximum rule value.

Some limits of the caSpec must be acknowledged. While an expanded caSpec can accommodate CA with more dimensions, the model presented here focuses only on one-dimensional CA. CA in two or more dimensions, however, may not be necessary to obtain a wide range of CA behaviors. As Stephen Wolfram has argued, CA of all dimensionality seem to exhibit the same types of behavior and are capable of the same levels of complexity (2002, p. 654). Furthermore, some two-dimensional CA, such as John Horton Conway's popular "Game of Life," when examined one dimension at a time, produce behavior nearly identical to that of a one-dimensional CA (Wolfram 2002, p. 245). Not only is one-dimensional data (once mapped or dynamically controlled) applicable to a wide range of musical parameters, but also a one-dimensional CA is computationally more efficient. Although a multi-dimensional CA may offer uniquely correlated data streams, multiple one-dimensional CA may be run in parallel, possibly producing behavior analogous to a multi-dimensional CA.

The model presented here, by defining `y` as a static parameter and specifying sub-table parameters, suggests that the CA table will not be generated in real time but rather calculated in advance of final deployment. This does not preclude the use of dynamic rule and mutation values, nor does it limit reading or mapping the generated CA values in real-time.

The caSpec and a compatible multi-format, one-dimensional CA are implemented in the Python module automata.py. This module is distributed as part of the cross platform and open source (General Public License) athenaCL libATH library. Complete implementation details and object-design analysis are beyond the scope of this study.

The caSpec in Use

To reiterate, the caSpec introduced here provides a platform-independent input notation for generating CA. A complete Python-based implementation of the caSpec is found within athenaCL. The athenaCL AUca command, for AthenaUtility Cellular Automata, permits testing and visually exploring a CA configured with a caSpec, dynamic mutation values, and dynamic rule values. This command will be used here to demonstrate CA parameters.

The AUca command requires three space-delimited arguments: a caSpec, a rule value or generator, and a mutation value or generator. As stated above, users may provide a caSpec with as few parameters as necessary and in any order. Rule and mutation arguments may be provided with static values or with athenaCL ParameterObjects. In athenaCL, ParameterObjects, as models of one-dimensional generators, offer high-level object interfaces to CAAC tools and procedures (Ariza 2005a, pp. 205–207).

Figure 1 illustrates a scenario in which the user has created a continuous CA with rule 0.48. The

Figure 1. A continuous CA.

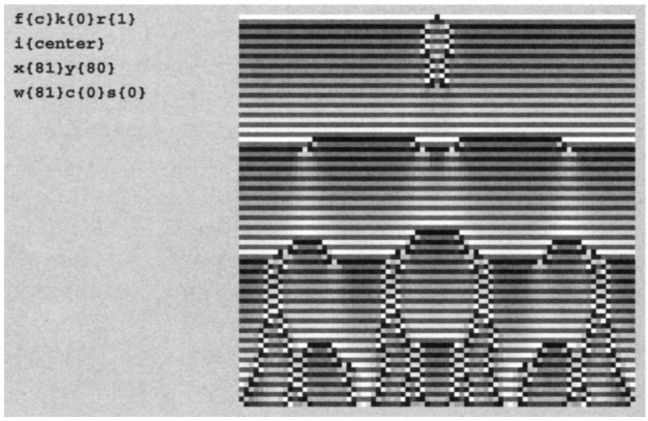


Figure 2. A float CA.

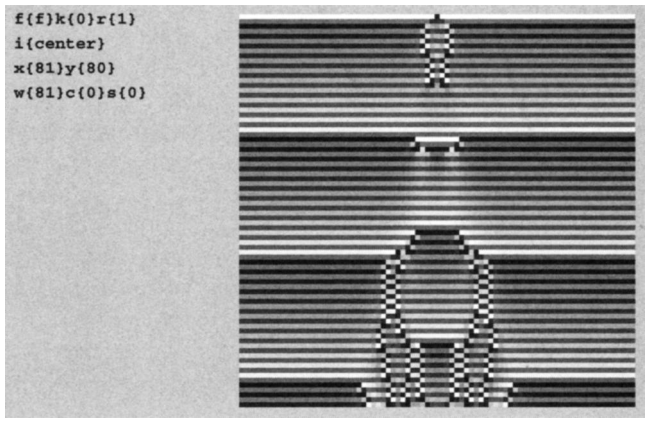


Figure 2

Figure 3. A float CA with skip and center parameters.

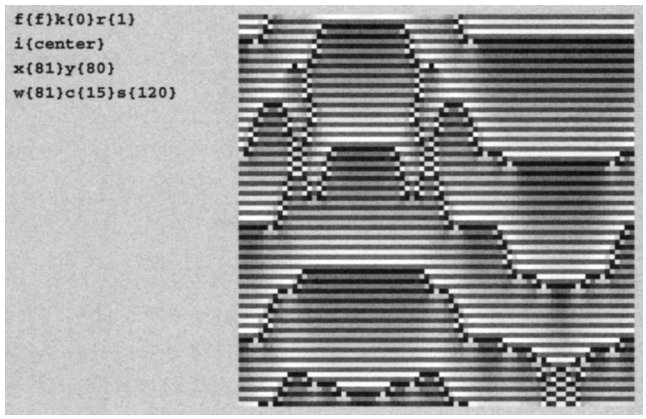


Figure 3

caSpec `f{c}x{81}y{80}` is supplied to the AUca command. The expanded caSpec, with default parameters supplied, is displayed in the left margin of the generated image. To meet the challenge of producing dynamic CA behavior within small cell state histories, CA presented here will use a table size of `x{81}y{80}`.

As mentioned herein, the difference between continuous and float CA is in the low-level implementation of cell value data types. To ensure accuracy, continuous CA cell state values use the Python 2.4 decimal data type (Cowlishaw 2005). Alternatively, standard Python floating-point data types may be used. Although floating-point data types offer better performance, in some cases accumulated floating-point error results in differences from "true" continuous CA. Whereas Wolfram dismisses these floating-point CA as "qualitatively wrong" (2002, p. 921), here they are embraced as an alternative continuous CA format. Figure 2, with a float CA, demonstrates the same rule as in Figure 1. The caSpec `f{f}x{81}y{80}` is supplied to the AUca command.

In Figure 3, the skip parameter is set to 120 to advance the displayed CA 120 rows (or generations) past the initial row. (Note that 200 total rows must be calculated; as specified by the `y` parameter, the last 80 rows are presented in the subtable.) The center parameter is set to 15, rotating the CA table 15 columns to the left. The caSpec `f{f}x{81}y{80}s{120}c{15}` is supplied to the AUca command.

The composition of the initial CA row is configured with the `i` parameter. Figure 4 maintains the previous settings but uses a randomly constructed initial row. The caSpec `f{f}x{81}y{80}s{120}c{15}i{r}` is supplied to the AUca command.

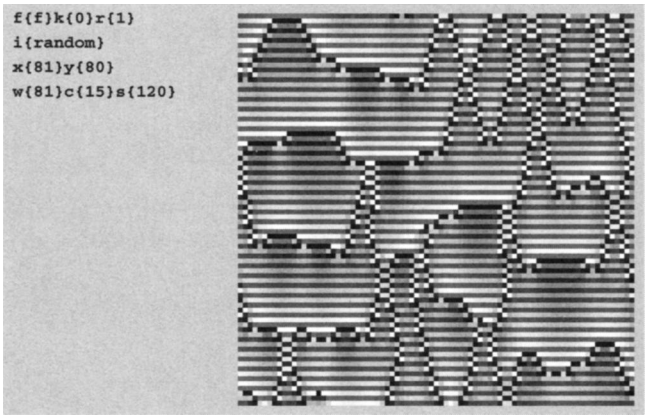
To isolate columns from the CA table, the width parameter in Figure 5 is set to 10. The initial row parameter, no longer specified, assumes the default value of `center`. The caSpec `f{f}x{81}y{80}s{120}c{15}w{10}` is supplied to the AUca command.

Owing to the effects of row edge-wrapping, subtables, isolated from a larger CA table, cannot be

Figure 4. A float CA with random initialization.

Figure 5. A float CA with skip, center, and width parameters.

Figure 6. A float CA with equal x and width parameters.



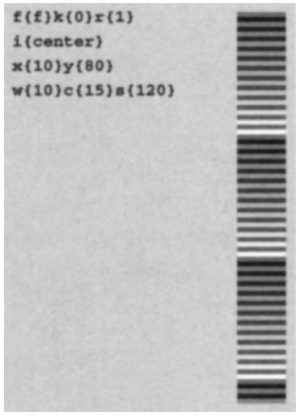


Figure 4

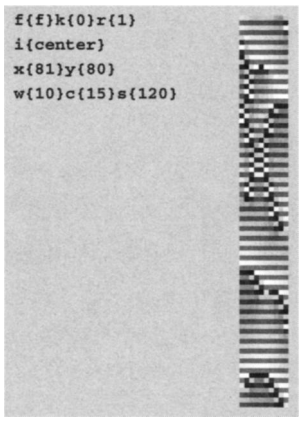


Figure 5

generated by simply creating a CA with an *x*-axis size equal to the desired sub-table width, in most cases. For example, the settings used in Figure 5 are demonstrated in Figure 6 with an *x*-axis size set to 10. A markedly different cell structure results. The caSpec `f{f}x{10}y{80}s{120}c{15}w{10}` is supplied to the AUca command.

An Overview of Historical Music Models

Now that the parameters of the CA and caSpec have been defined and demonstrated, a review of the historical applications of CA in music, with attention to specific CA models and applications, can be provided. Significant research in employing CA for the generation of musical structures began in the late 1980s. Recently, an overview of some of this work has been supplied by Burraston and others (Burraston et al. 2004; Burraston 2005, 2006; Burraston and Edmonds 2005). Jacques Chareyron and Peter Beyls demonstrated some of the earliest musical research (Chareyron 1988a, 1988b, 1990; Beyls 1989, 1990a, 1990b, 1991a, 1991b, 2003, 2004).

Chareyron demonstrates the use of one-dimensional CA (or "linear automata") for directly specifying waveforms. Chareyron relates this technique, a type of non-standard synthesis, to the Karplus-Strong algorithm (1983): the waveform is "self-modifying" in that each previous waveform, as a cellular lattice, is processed to produce the next waveform (Chareyron 1990, p. 27). Although Chareyron suggests that "in musical applications it may be a good idea to add small random modifications to the wavetables to allow the repetition of similar but not identical tones" (1990, p. 30), the specific application of this randomness is neither specified nor demonstrated in the supplied code.

Beyls, working on an Atari computer in the Mach2 Forth language, began exploring applications of CA to music in the early 1980s (1989, 2006). This work was based in part on visual CA-based animation experiments conducted during 1978–1980 (2006). Beyls's research is part of a broader investi-

gation into the emergent properties of dynamical systems, and his CA models extend standard formulations with feedback and multi-level automata (1989, p. 40), applications of multiple parallel CA and probabilistic cell-state "aging" processes (1991b, p. 255), and integration with genetic algorithms and use of CA cell-state mutation or "noise" (2003). While this research is diverse, practical and portable implementations of Beyls's tools are not readily available.

Much additional research followed that of Chareyron and Beyls. Battista and Giri employed CA for generating MIDI note events (1988). Bowcott (1989), in an early application of CA values to synthesis parameters, demonstrated the use of CA-generated values for granular synthesis. Dave Millen demonstrated MIDI-based applications of one-dimensional CA to pitch and duration structures (1990, 1992, 2004). Millen mentions, but does not demonstrate, the utility of inserting "new 'initial' cell state values or . . . changing . . . existing state values" (1990, p. 316). Although this work is related to automata-bending techniques, it is not clear whether Millen makes these alterations during or after CA processing. Eduardo Reck Miranda's CAMUS (and CAMUS 3D) software employs a fixed mapping of CA cell positions to trichord intervals for MIDI-based music production (Miranda 1993, 2003; McAlpine et al. 1999). The Chaosynth system, also by Miranda, offers an alternative application of CA to granular synthesis (Miranda 1995, 2003; McAlpine et al. 1999; Miranda et al. 2000). Additional applications of CA to music are demonstrated by Hunt et al. (1991), Comajuncosas (1998), Elliott (1998), Bilotta and Pantano (2002), Dorin (2002), Eldridge (2002), Reiners (2004), Burdick et al. (2004), and Karaca (2005).

As Burraston et al. (2004) illustrate, the majority of CA systems, excluding those dedicated to synthesis applications, have focused on mapping CA values to pitch and duration (2004, p. 76). This is in part because CA values are often hard-wired to system-defined mappings, such as the fixed translation of CA cell positions to interval distances in Miranda's CAMUS. Recent research by Burraston (2005, 2006), as well as the availability of CA-based tools in synthesis envrionments such as Kyma (Scaletti 1987), demonstrate applications of CA that are more modular and that are applied to more diverse musical parameters.

This research likely represents only a fraction of the musical explorations of CA to music. Joel Ryan, with artist Ray Edgar, employed a CA for both sound and image in a multi-media installation entitled *LINA* (1985; Beyls 1989, p. 37; Ryan 1991). Iannis Xenakis, although perhaps more inclined toward less deterministic procedures (Xenakis 1992), describes the use of a CA in the composition of *Horos* (1986). In 1989, Xenakis remarked that, apart from work on his sieves (Xenakis 1990; Ariza 2005b), "the only new procedure I've used is the so-called cellular automata" (Varga 1996, p. 199). Xenakis mapped CA table columns to a 23-element pitch scale, and he used contiguous sections of cell activity to determine pitch structures (Hoffman 1993, 2002; Varga 1996, p. 200). Peter Hoffman summarizes this mapping with a metaphor: "it is as if a collection of tuned chimes were rung by a complex mechanism" (2002, p. 126). Hoffman (2002) furthermore identifies CA-derived structures from *Horos* in Xenakis's *Ata* (1987).

Referencing applications of CA to fluid dynamics, Xenakis states that his techniques create musical structures where "the sound is a kind of fluid in time" (Varga 1996, p. 200). Hoffman, suggesting much more, states that Xenakis's use of CA can be "understood as an implicit demonstration of the strength and limitation of universal computation in music composition" (2002, p. 124). First, it is not certain that the particular totalistic CA used in *Horos* is capable of universal computation. Second, although Xenakis does not employ automata-bending techniques, Hoffman documents how Xenakis manipulates the output of the CA to avoid repetition and to "perturb the CA's lateral symmetry" and, furthermore, how Xenakis includes numerous arbitrary alterations Hoffman calls "errors" (2002, p. 126). As such, it is difficult to understand how Xenakis's use of CA demonstrate anything at all about universal computation. Instead, Xenakis's methods suggest that he found the "pure" output of the CA insufficient for musical application.

Figure 7. A standard CA without mutation.

Figure 8. A standard CA with mutation.

Automata Bending with Mutation and Dynamic Rules

Despite the abundance of research exploring applications of CA to music, few studies have incorporated automata-bending techniques. As stated previously, two methods are demonstrated here: mutation and dynamic rules. Mutation is specified as a probability, within the unit interval, of bypassing conventional rule application. If mutation is activated, a cell's state is randomly selected from within the set or range of possible cell states. Each cell in a row is a potential candidate for mutation. A single mutation probability is applied to each cell in a row; if a dynamic mutation value is provided, this probability is only updated once per row. Dynamic rules permit altering the rule value used in the generation of each row of the CA. As with mutation, the rule value may only change once per row. As stated earlier, in the case of discrete CA, fractional rule values are treated as probabilistic weightings toward nearest integers, and rule values beyond the system's maximum rule value are resolved by a modulus operation.

In Figures 7 and 8, two standard-format (`k{2}r{1}`) CA are presented. In Figure 7, the CA demonstrates repetitive behavior common to many CA. In Figure 8, random mutation at a constant rate of 0.003 is employed. By introducing mutation, a repetitive pattern is transformed into a heterogeneous collection of forms; the fixed CA rule, however, constrains the resulting forms within a small collection. While the specific arrangement is random, three basic forms are found: open vertical columns, horizontal bracket-like shapes of various widths, and checkered patterns. The complete AUca command, following the athenaCL prompt (`::`), is given before each image in Figure 7 and Figure 8.

Application of mutation is probabilistic and thus breaks the determinacy of a traditional CA. In Figures 9 and 10, two CA with identical caSpecs are both given a constant rule value (1842) and a mutation value that, by use of the athenaCL BreakPoint-Linear ParameterObject, linearly increases from 0.00 to 0.02. Although the input in each case is identical, probabilistic mutation creates unique

```
:: auca f{s}x{81}y{80}k{2}r{1} 109 0
```

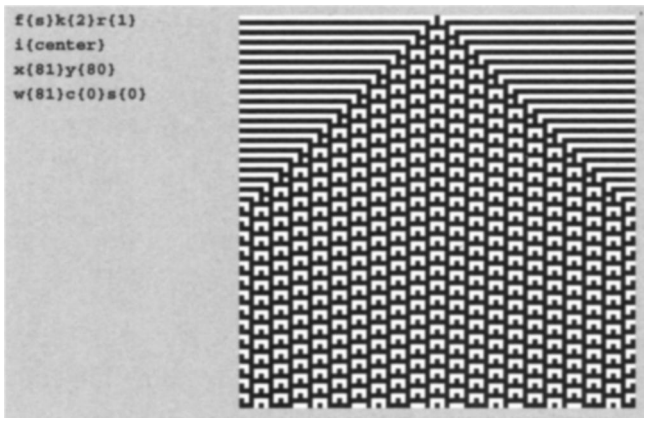


Figure 7

```
:: auca f{s}x{81}y{80}k{2}r{1} 109 0.003
```

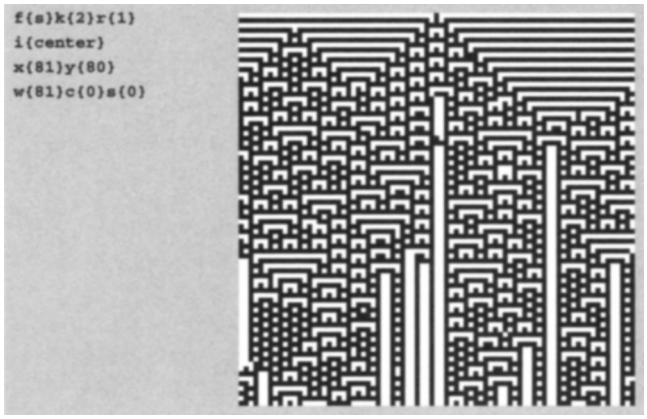


Figure 8

outputs. Further, the increase in mutation probability, combined with the accumulated cell formations, produces a directed change in the CA behavior.

The application of dynamic and probabilistic rules permits the generation of CA that escape the singular behaviors of homogeneity, repetition, randomness, or complexity (Wolfram 1983), instead offering heterogeneous and dynamic behavior. As mentioned previously, fractional rule values in discrete CA are treated as probabilistic weightings toward nearest integers. In Figure 11, a standard-

Figure 9. A totalistic CA with linearly increasing mutation.

Figure 10. A totalistic CA with linearly increasing mutation.

Figure 11. A standard CA with probabilistic rule 90.5.

Figure 12. A standard CA with rules 90 and 91.

```
:: auca f{t}x{81}y{80}k{3}r{1} 1842
  bpl,e,l,((0,0),(80,.02))
```

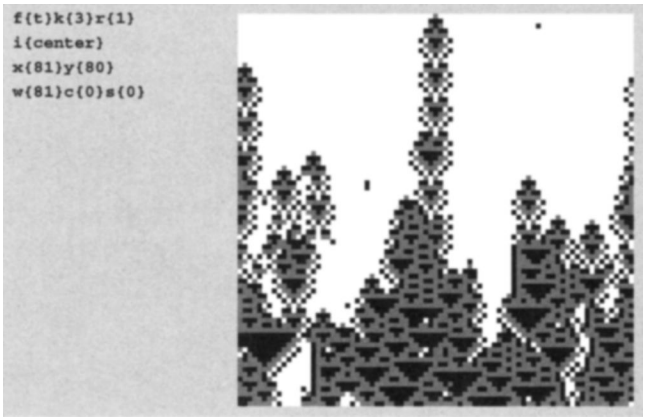


Figure 9

```
:: auca f{t}x{81}y{80}k{3}r{1} 1842
  bpl,e,l,((0,0),(80,.02))
```

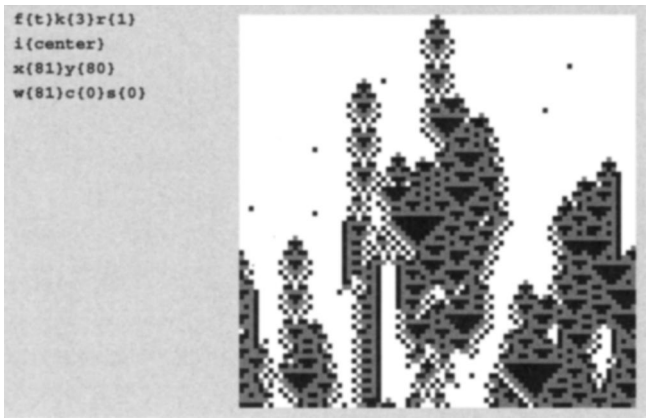


Figure 10

```
:: auca f{s}x{81}y{80}k{2}r{1}i{r} 90.5 0
```

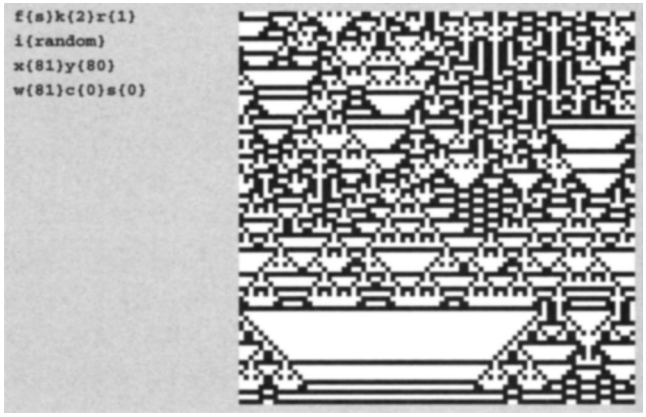


Figure 11

```
:: auca f{s}x{81}y{80}k{2}r{1}i{r}
  wp,e,80,.5,90,91 0
```

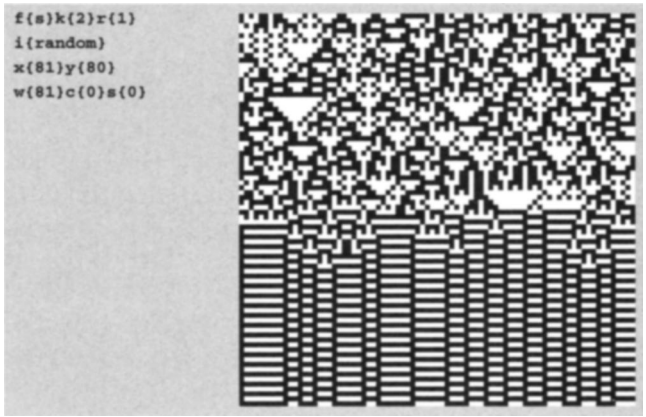


Figure 12

format (`k{2}r{1}`) CA is created with a constant rule of 90.5. Rule 90, started from `i{center}`, is well known for producing a discrete analogue of the Sierpinski gasket fractal; started from `i{random}`, rule 90 produces noisy data with occasional triangular formations. Rule 91, started from either `i{center}` or `i{random}`, produces repetitive, static behavior. A probabilistic rule of 90.5, as demonstrated in Figure 11, produces a heterogeneous combination of behaviors from rule 90 and rule 91. As initial conditions, as well as rule application, are probabilistic, each generation of the CA produces unique results. For comparison, Figure 12 employs the athenaCL WavePulse ParameterObject to split 80 steps between rules 90 and 91.

Although the numbering of CA rules does not directly correlate to CA behavior, smoothly changing rule values over the course of a CA evolution pro-

Figure 13. A standard CA with a linearly increasing rule from 30 to 34.

```
:: auca f{s}x{81}y{80}k{2}r{1}i{r}
  bpl,e,s,((0,30),(80,34)) 0.05
```

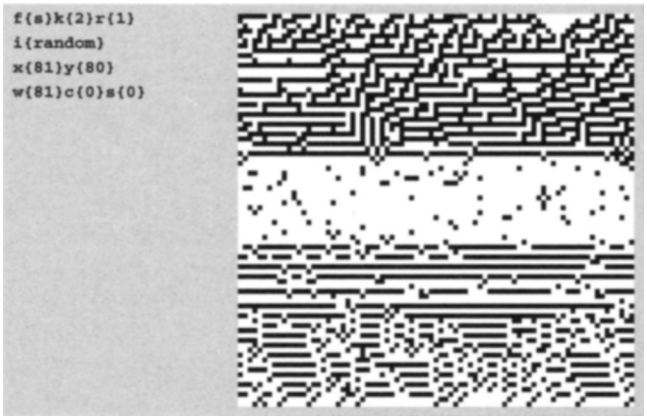
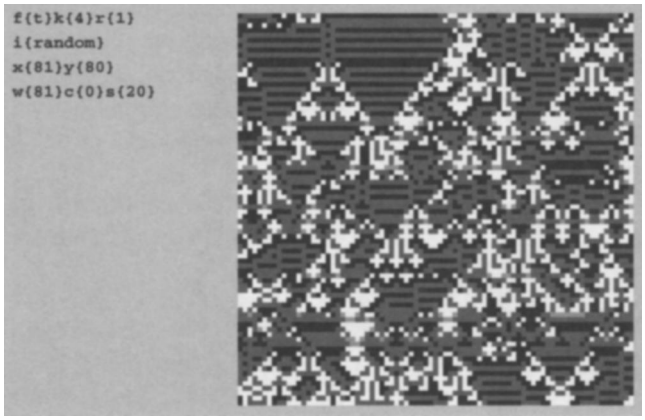


Figure 14. A totalistic CA with rules supplied by a zero-order Markov chain.

```
:: auca f{t}y{80}x{81}r{1}k{4}i{r}s{20}
  mv,a{195735784}b{846484}:{a=3|b=1} 0
```



duces directed behaviors. The probabilistic weighting of floating-point rule values, in most cases, promotes these gradual transitions by mixing the effects of two rules. In some cases, dynamic rules may result in the cessation of CA activity; small amounts of mutation can be added to mitigate this problem. In Figure 13, a standard-format (`k{2}r{1}`) CA is created with a linearly increasing rule from 30 to 34 and a constant mutation of 0.05. Although rule 30 is well known for its random behavior (Wolfram 1985), in this case rule 30 is used as just one of four transitional behaviors. The boundaries between these behaviors are necessarily fuzzy, at least four regions can be identified in Figure 13: triangular shapes, open space, horizontal rows, and diagonal lines. In this case, mutation, visible as the random cell activity in the open-space region, is necessary to prevent the cessation of CA activity.

Rather than a smooth, linear transition of rule values, a variety of alternative generators can be used to supply rule values. In Figure 14, the athenaCL MarkovValue ParameterObject (Ariza 2006) employs a zero-order Markov chain to probabilistically supply a totalistic, `k{4}r{1}` CA rule 195735784 three times more frequently than rule 846484. A skip value of 20 is used to extract values after the immediate effects of the random initial conditions. The resulting CA demonstrates two intermingled behaviors.

Extracting Table Values into One-Dimensional Sequences

Given a table generated by a one-dimensional CA, the process of converting this tabular data into numerical values and then mapping these numerical values to musical parameters can be approached in many ways. A common technique extracts two-dimensional data from active (non-zero) cells: table columns are treated as discrete pitch values, or a scale, and table rows are treated as discrete time steps. An active cell is interpreted as an event with a pitch specified by the cell's column; this pitch is sustained for as many rows as the specific column is active. This direct pitch-to-column, duration-to-row mapping is employed by Xenakis (Varga 1996, p. 200) and others. Direct two-dimensional mapping techniques, however, often result in a fixed interpretation of column and row values and do not permit more statistical and general interpretations of CA behavior.

Converting a table (or a sub-table) generated by a one-dimensional CA into a one-dimensional value sequence offers a flexible alternative. The most literal method is to concatenate all cell state values into a single sequence. This "flattening" of the table can proceed from row to row, or from column to column, and can proceed in forward or reverse, per row or column. Concatenating table column values,

Table 1. Table Extraction Parameters (an Asterisk Designates a Default Parameter)

Parameters				*Methods*	
Type	*Axis*	*Source*	*Filter*	*Count*	*Examples*
flat	row column rowReflect columnReflect	value* index passive	none* active	24	flatRowActive flatRowReflectIndexActive flatColumnIndex flatColumnReflectPassive
sum product average	row column	value* index	none* active passive	36	sumRow productColumnIndexPassive averageRowActive

rather than row values, may operate against the linear evolution of the CA; yet in the context of artistic application, there is no mandate to preserve the directed evolution of the CA.

Alternatively, cell row index positions (column numbers), rather than cell values, can be extracted; without additional filtering, however, these values are not of particular use. Table values may be filtered by only extracting cells that are either active (not equal to zero) or are passive (equal to zero). As the value zero in a CA has no absolute meaning, such a filtering mechanism is arbitrary, and is only unbiased in the case of a binary `k{2}` CA. Musical applications of CA data, particularly those employing `k{2}` CA, have frequently defined active cells as more meaningful than passive cells. By employing a filter, a one-dimensional sequence can be created by concatenating only active or only passive cell values or indices. Collecting all active cell indices is closely related to the two-dimensional mapping procedure described earlier, where indices of active cells map to a discrete parameter such as pitch.

Rather than being concatenated, the sequence of values in rows or columns can be processed with various arithmetic operations to return a single value, and this value can be used to build a sequence of CA-derived values. Row- or column-cell values can be added, averaged, or multiplied. Row or column indices may also be similarly processed. As before, index values become more useful when filtered; filtering may be applied while extracting index values or cell values.

There are then four parameters available to configure one-dimensional sequence extraction from a CA table or sub-table: (1) *type:* flat, sum, product, or average; (2) *axis:* row or column, and (if the extraction type is flat) row reflect or column reflect (where reflection designates reversing each row or each column before concatenation); (3) *source:* value or index; and (4) *filter:* none, active, or passive. For flat extraction types, there are six possibilities for each of four axis values (24 total); for sum, product, and average extraction types, there are six possibilities for each of two axis values (36 total). Table 1 shows these parameters. Although these extraction methods are too numerous to demonstrate, it can be shown that, even for elementary CA tables, each of the 60 methods may offer a unique value sequence.

Even though various extraction and mapping techniques may offer diverse numerical interpretations of CA data, the resulting musical structures, in most cases, can be seen only as raw musical materials. It is always the composer that provides final musical interpretation and aesthetic integration of computer-generated materials (Koenig 1983).

One-Dimensional CA Sequence Generation in athenaCL

The previous discussions of the caSpec, the application of dynamic mutation and dynamic CA rules, and the methods of table value extraction, offer language-

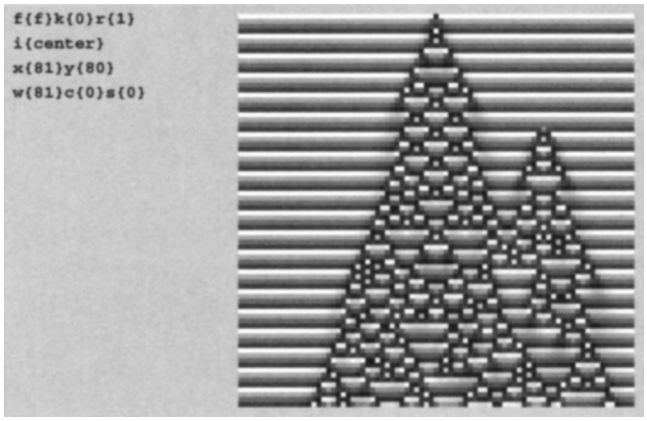


Figure 15. A float CA with a static rule of 0.25 and a static mutation of 0.0005.

and system-independent approaches to working with CA and CA-generated values. This section demonstrates the use of athenaCL ParameterObjects for one-dimensional CA sequence generation. As stated herein, ParameterObjects are models of one-dimensional generators. ParameterObjects can be supplied as arguments to other ParameterObjects, permitting complex, embedded dynamic generators. At a higher level of system organization, musical parts in athenaCL are deployed as specialized TextureModules, or multi-dimensional generators (Ariza 2005a, p. 227); each Texture is configured with the assignment of many ParameterObjects.

Within athenaCL, the method of extracting values from a CA table is specified with a single string argument formed from the parameters defined earlier and illustrated in Table 1. To make these strings compact, the source parameter `value` is assumed (unless `index` is specified), and the filter `none` is assumed (unless `active` or `passive` is specified). The athenaCL system supports automatic acronym expansion for all string arguments (Ariza 2005a, p. 242). As all 60 table extraction methods have unique acronyms, this feature provides easy user access to the many varieties of extraction methods. The string `fcr`, for example, can be provided to specify the `flatColumnReflect` method; the string `aria` can be provided to specify the `averageRowIndexActive` method.

Two ParameterObjects are provided for generating numeric values with CA: CaList and CaValue. CaList takes six arguments: (1) name, (2) caSpec, (3) parameterObject (rule), (4) parameterObject (mutation), (5) tableExtractionString, and (6) selectionString. The name of a ParameterObject must be provided as a first argument. A caSpec is provided as a second argument. The third argument is an embedded ParameterObject to supply the CA rule value. The fourth argument is an embedded ParameterObject to supply the CA mutation probability. The fifth argument is a string to specify the extracting method. As ParameterObjects are designed to continually produce values, the sixth argument provides a selection string to determine how the CA values, extracted from the table, are ultimately deployed. Valid selection strings are as follows: `randomChoice`, `randomWalk`, `randomPermutate`, `orderedCyclic`, and `orderedOscillate`.

The CaValue ParameterObject is nearly identical to CaList, except that all table values, after extraction, are normalized within the unit interval. These normalized values are then scaled within a dynamic range defined by two ParameterObjects (`min` and `max`). CA-generated values can thus be shaped by what Koenig called a *tendency mask* (1970). CaValue takes eight arguments: (1) name, (2) caSpec, (3) parameterObject (rule), (4) parameterObject (mutation), (5) tableExtractionString, (6) min, (7) max, and (8) selectionString.

These ParameterObjects will be used to demonstrate a few of the many table-value extraction methods. A float CA with a static rule of 0.25 and a static mutation of 0.0005, shown in Figure 15, is used to generate value sequences in Figures 16, 17, and 19. These event-domain graphs map event step (*x*-axis) to event (*y*-axis) value and provide a visual display of one-dimensional values prior to musical mapping and interpretation.

Figure 16 employs the sum of CA table rows, and Figure 17 is based on the sum of CA columns. In both cases the resulting parameter values demonstrate complex, directed motion with partial branching. In both cases extracted value sequences will be repeated due to the use of the orderedCyclic selection method; as there are only 81 column-derived values, this periodicity is only visible in Figure 17.

CA data, once extracted, can be mapped to musi-

Figure 16. Table values extracted as rows that have been added together.

Figure 17. Table values extracted as columns that have been added together.

```
caList, f{f}k{0}r{1}i{center}x{81}y{120}
  w{80}c{0}s{0}, (constant, 0.25), (constant,
  0.0005), sumRow, orderedCyclic
```

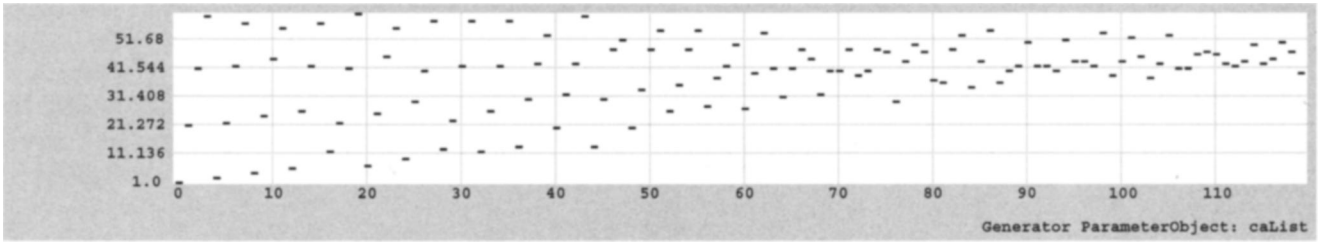


Figure 16

```
caList, f{f}k{0}r{1}i{center}x{81}y{120}
  w{80}c{0}s{0}, (constant, 0.25), (constant,
  0.0005), sumColumn, orderedCyclic
```

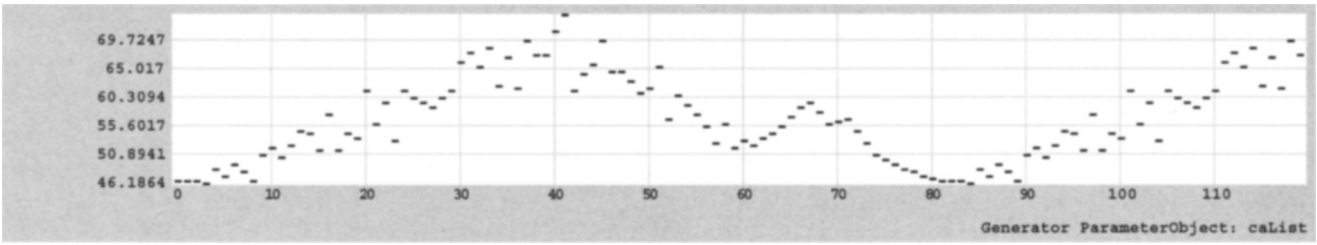


Figure 17

cal parameters with any procedure used for similar one-dimensional value sequences, including procedures that employ dynamic or non-deterministic mappings. In the following examples, simple, direct musical mappings are provided to demonstrate the many possible applications of CA-derived value streams. These musical structures are intentionally elementary and are not demonstrations of composed musical examples. Note that, owing to the indeterminacy of bent automata generators, the data illustrated in event-domain graphs is not identical to the data used to produce musical mappings.

A direct musical mapping to pitch of values produced in Figure 16 can be accomplished within athenaCL by assigning the CaList generator to the field parameter of TextureModule LineGroove. The field parameter transposes event pitch values, here based on a fixed, Path-supplied (Ariza 2005a, p. 152) pitch of C2 (where C4 denotes middle C). Floating-point, microtonal pitch specifications are rounded to the nearest equal-tempered pitch, in this case. Given a rhythm generator that produces permutations of four Pulses (three sixteenth notes and one dotted eighth note), the resulting arpeggios place durational emphasis on varying pitches within the narrowing four-note arpeggios. The MIDI output, produced with athenaCL, is transcribed in Figure 18.

Figure 19, employing the float CA in Figure 15, demonstrates the utility of using the width parameter to read a small range of columns from a CA. Values are derived by extracting flat rows from the CA sub-table. Repeated sub-sequences can be found, though sometimes these sub-sequences are slightly shifted or reversed and interspersed with unique, developing formations.

Figure 18. Transcribed CA-generated pitch values.

Figure 19. Masked sub-table values extracted as flat rows.

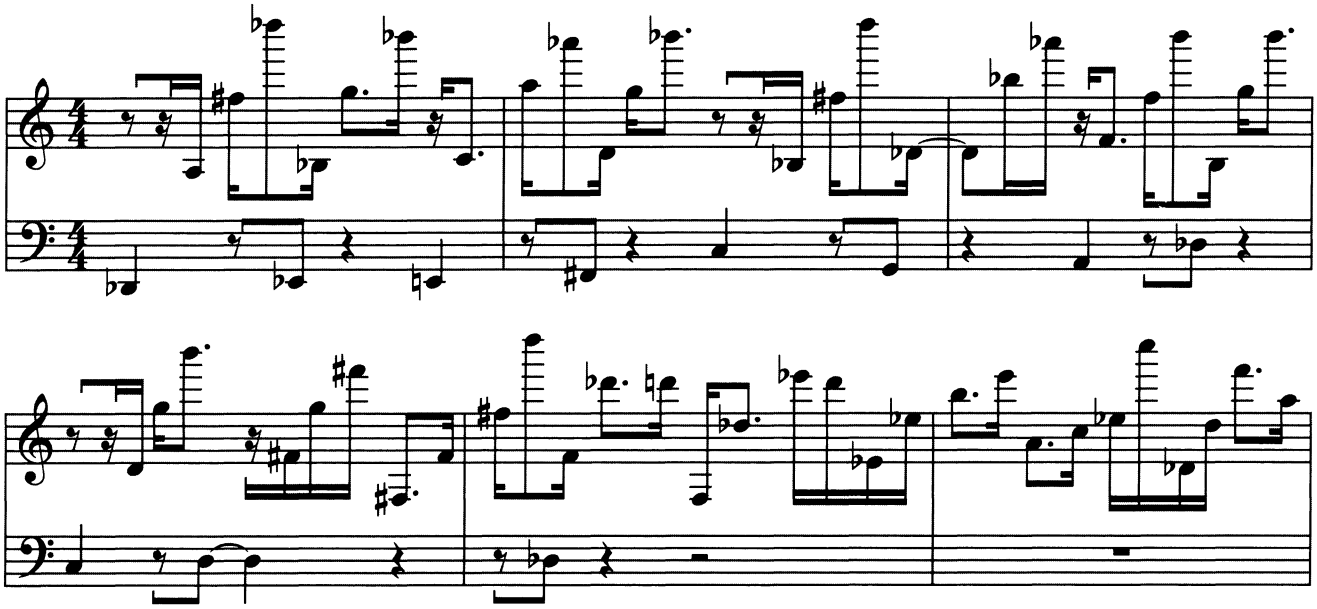


Figure 18

```
caList, f{f}k{0}r{1}i{center}x{81}y{120}
  w{10}c{0}s{20}, (constant, 0.25), (constant,
  0.0005), flatRow, orderedCyclic
```

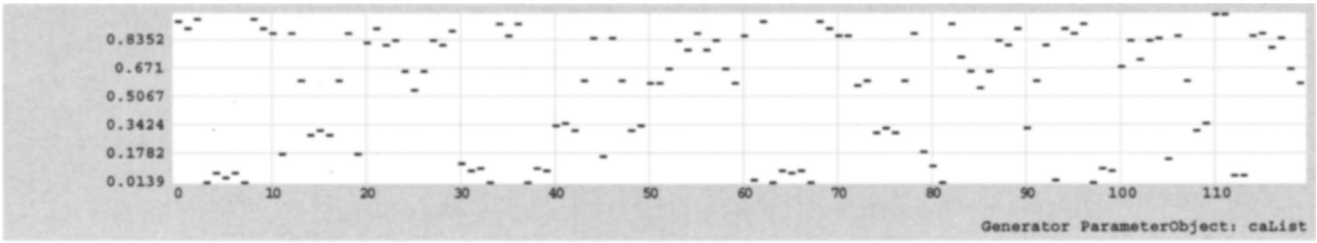


Figure 19

The floating-point values obtained in Figure 19 can be applied to a wide range of musical parameters. For example, within athenaCL the CaList generator can be applied to an auxiliary parameter to control a Csound (Vercoe 1986) sample-playback instrument. In the production of Figure 20, a CaList ParameterObject, configured as in Figure 19, is assigned to the initial playback speed of a single tone steel-drum audio sample. The final playback speed is set at a constant value of one. The result is an increasing playback speed between the CA-derived value and one over the duration of the event. Uniform envelopes, a linearly increasing tempo, and random permutations of the same rhythms used in Figure 18 are employed. Figure 20 (produced with the open-source audio editor Audacity) shows a spectral analysis of 20 seconds of audio, using a Fast Fourier Transform window of 1,024 points and displaying frequencies from 43 to 8,000 Hz.

By normalizing accumulated table values, the CaValue ParameterObject offers a numerically uniform presentation of values produced by diverse CA

Figure 20. Spectral analysis of audio employing dynamic, CA-generated sample playback speeds.

Figure 21. CA values scaled by an embedded ParameterObject.

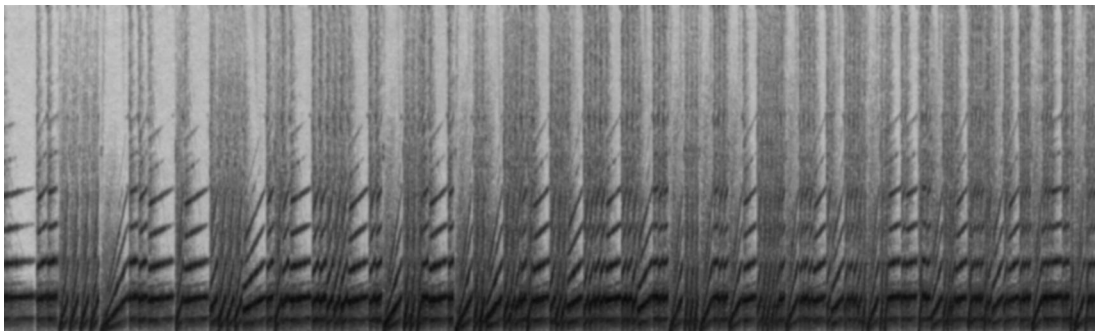


Figure 20

```
caValue, f{t}k{3}r{1}i{center}x{81}y{120}
  w{12}c{0}s{0}, (constant, 1842),
  (breakPointLinear, event, loop, ((0,0),
  (80,0.02))), sumRow, (waveSine, event, 15,
  0, (constant, 0), (constant, 0.4)),
  (constant, 1), orderedCyclic
```

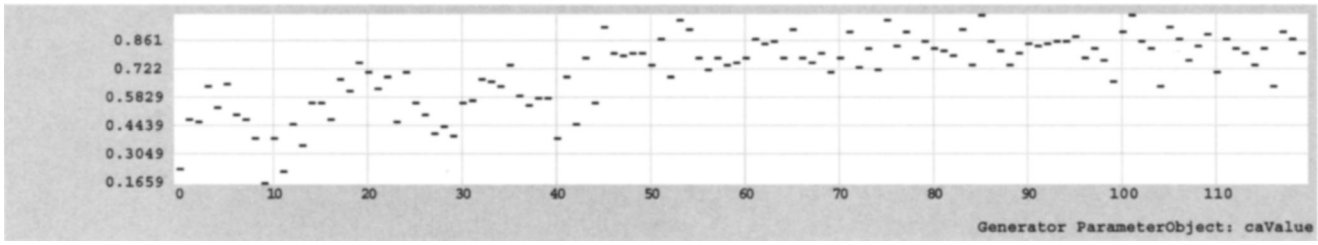


Figure 21

formats. Figure 21 employs the CA demonstrated in Figure 9 within CaValue; when rows are added together and extracted from the table, the increasing cell density, accelerated by the increasing mutation, causes a gradual but punctuated increase in values. These values are given a slight oscillation with a WaveSine ParameterObject operating as a lower boundary.

Figure 22 uses the CA demonstrated in Figure 13; even when presented as the values of rows that have been added together, the gradual transition between rules is apparent in the diverse value sequence behavior. Here, the BreakPointLinear ParameterObject extends the transition of rule values, from 30–34, over 120 steps. The `min` and `max` are set to constant values of 100 and 8,000, respectively.

Two CaValue generators, both configured as in Figure 22, can be applied to two auxiliary Texture parameters in athenaCL to control a Csound filtered-noise instrument. In this case, the generators are applied to the start and end center frequencies of a band-pass filter. The center frequency thus varies over the duration of the event. Uniform envelopes, a constant 200 beats-per-minute (BPM) tempo, and random permutations of the same rhythms used in Figure 18 are employed. Figure 23 provides a spectral analysis, under the same conditions as Figure 20.

If the nested, Sierpinski gasket-like patterns of some CA (such as the `f{s}k{2}r{1}` rule 90 CA) are desired in a one-dimensional value sequence, the `flatRowIndexActive` extraction method may be used. In Figure 24, two CA rules that produce nested results, rule 90 and rule 182, are probabilistically in-

Figure 22. CA values demonstrating dynamic rules.

Figure 23. Spectral analysis of audio employing dynamic, CA-generated band-pass filter center frequencies.

Figure 24. Active index values extracted from a probabilistic CA.

```
caValue, f{s}k{2}r{1}i{random}x{81}y{120}
  w{81}c{0}s{0}, (breakPointLinear, event,
  single, ((0,30),(119,34))), (constant,
  0.05), sumRow, (constant, 100), (constant,
  8000), orderedCyclic
```

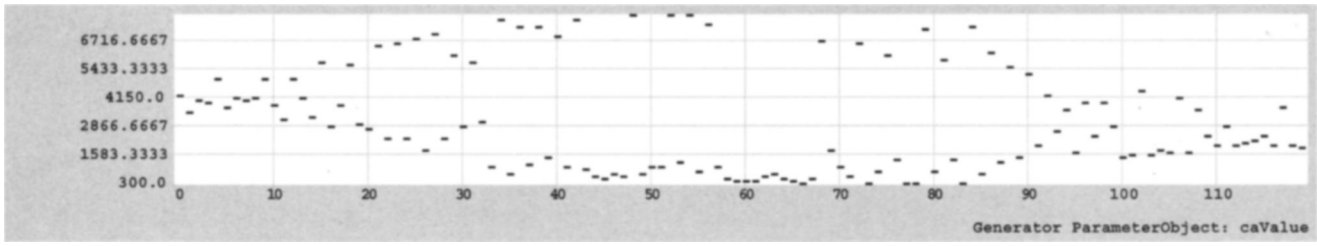


Figure 22

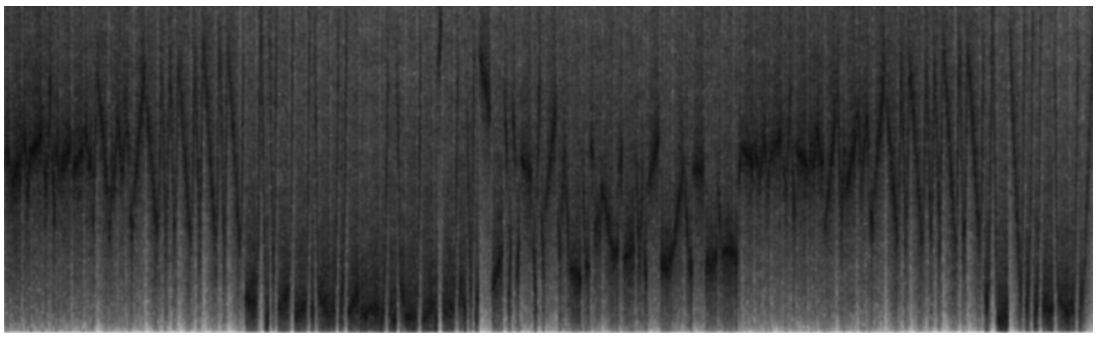


Figure 23

```
caValue, f{s}k{2}r{1}i{center}x{91}y{120}
  w{91}c{0}s{0}, (markovValue, a{90}b{182}:
  {a=29|b=1}, (constant, 0)), (constant, 0),
  flatRowIndexActive, (constant, 0), (constant,
  1), orderedCyclic
```

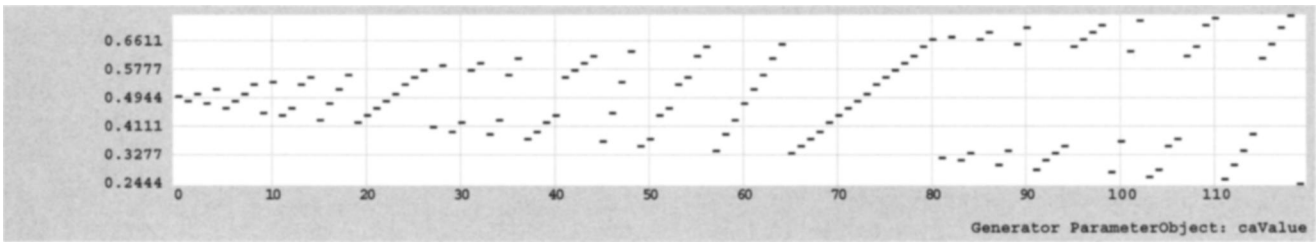
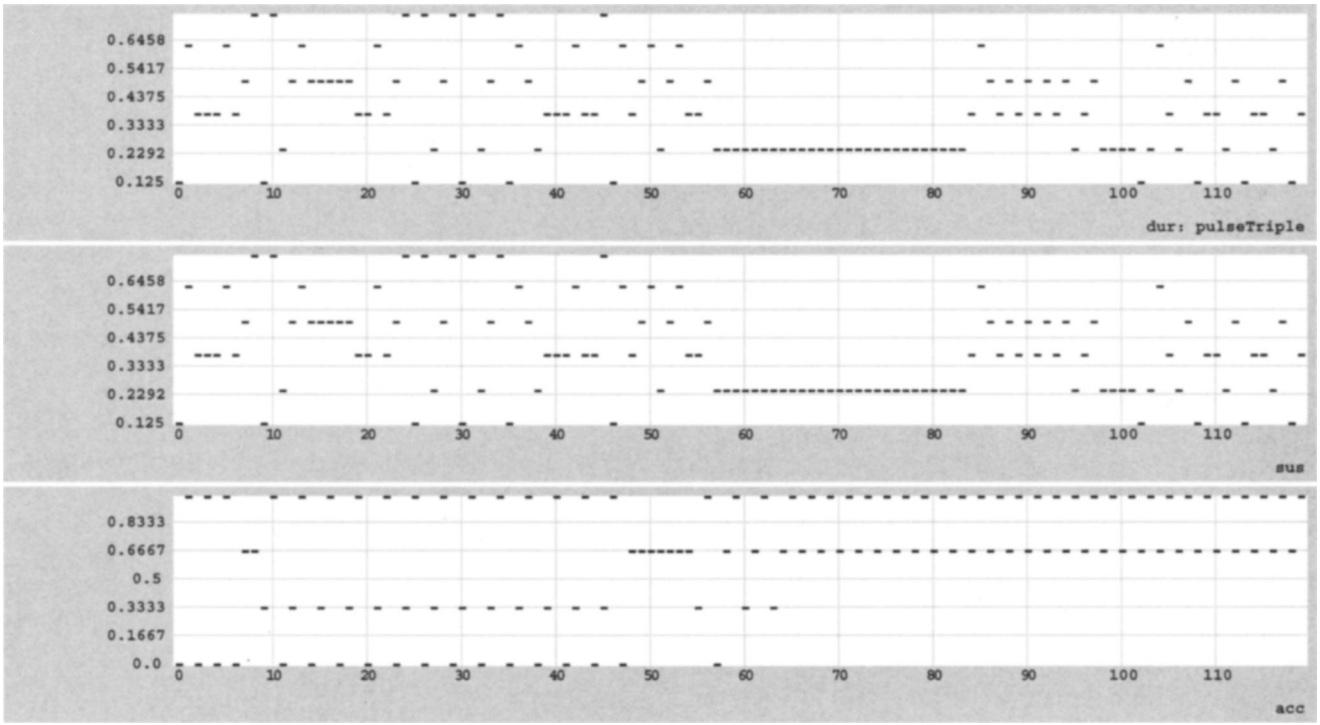


Figure 24

Figure 25. Rhythm values produced by two embedded CA, displaying duration, sustain, and accent for each event.

```
pulseTriple, (constant, 4), (caList, f{s}
  k{2}r{1}i{center}x{81}y{120}w{6}c{-2}s{0},
  (constant, 109), (constant, 0.01), sumRow,
  orderedCyclic), (caValue, f{s}k{2}r{1}
  i{center}x{81}y{120}w{3}c{8}s{0},
  (constant, 109), (constant, 0.003), sumRow,
  (constant, 0), (constant, 1),
  orderedCyclic), (constant, 1)
```



termingled. Rule selection is again controlled by a zero-order Markov chain, weighted strongly toward rule 90. CaValue is used here with `min` and `max` set to constant values of 0 and 1, respectively.

The athenaCL system features specialized ParameterObjects for rhythm generation. These generators employ Pulses—objects that encode a duration (as a ratio applied to an external tempo) and an accent (as an amplitude scalar between 0 and 1, or between a rest and a fully sounding event). The input notation for a Pulse is a Python list of three elements: a divisor, a multiplier, and an accent (Ariza 2005a, p. 163). Pulses may be dynamically constructed with Rhythm ParameterObjects such as PulseTriple. By employing this Rhythm ParameterObject with embedded CaList and CaValue ParameterObjects, diverse temporal structures can be created with CA-derived values.

Figure 25 uses the PulseList ParameterObject with two embedded CA generators to construct Pulse objects, each ParameterObject producing values from different presentations of the CA demonstrated in Figure 8. All pulse divisors are given a value of four, resulting in a common sixteenth-note subdivision. Pulse multipliers are produced with CaList, taking rows of the CA that have been added

Figure 26. Transcribed CA-generated rhythm values with mutation.

Figure 27. Transcribed CA-generated rhythm values without mutation.

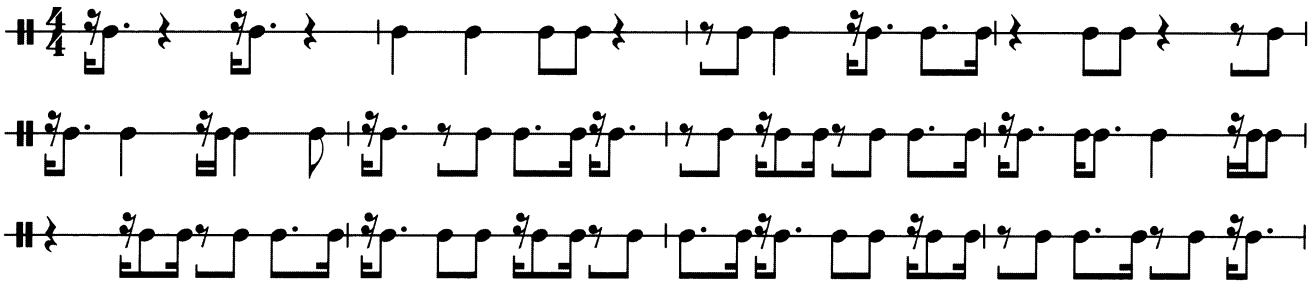


Figure 26

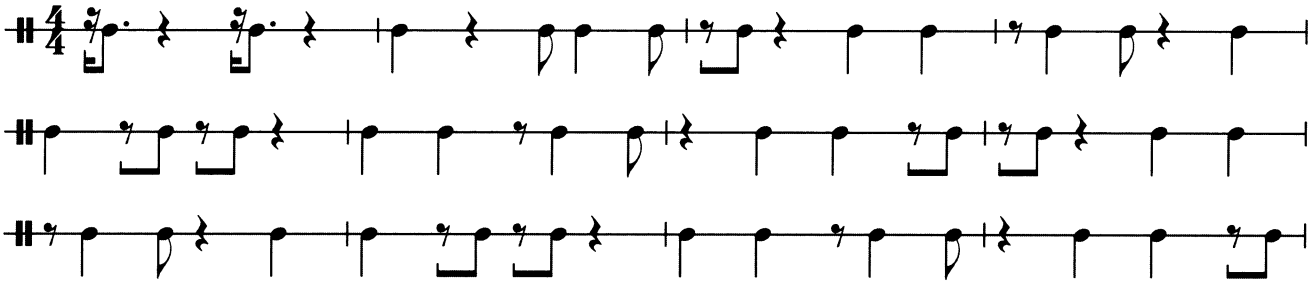


Figure 27

together with a shifted width of six. Seven possible integer values are thus possible. Pulse accent values are produced with CaValue, taking cumulative rows of the CA with a shifted width of three. Four possible floating-point values within the unit interval are thus possible. Figure 25 provides temporal values from generated Pulses, where calculated duration (time until next event), sustain (duration of event), and accent values are given in reference to a constant tempo of 120 BPM. A segment of a rhythm generated with the settings demonstrated in Figure 25 is transcribed, from athenaCL MIDI output, in Figure 26. The multiple levels of dynamic accent are not notated.

For comparison, the mutation values employed in Figures 25 and 26 can be set to zero. Retaining all other settings, including the same CA, an alternative rhythm is produced. Although no aesthetic merit can be applied to these examples devoid of context, the "un-bent" example provided in Figure 27 offers a more homogenous and nearly static rhythm.

Conclusion

Automata bending is an approach to manipulating the operation of a CA. With such techniques, CA rules that produce unexceptional behavior can be invigorated and made into more useful generators. Because CA with repetitive or random behavior seem to form the majority of all CA, these bending techniques may make more CA useful for creative applications. Some unbent automata may offer great creative utility, but in all cases, and for all rules, automata-bending techniques provide new types of parametric control, permitting a wide range of generators to influence or directly control CA evolution. The aesthetic results of this approach cannot be directly measured; as stated above, final musical interpretation and aesthetic integration are dependent on the composer and may be independent of any particular generative technique.

Early excitement about various dynamical systems led some researchers to herald a shift in the way computers could be used for composition. In

1991, Beyls stated that "complex dynamic systems are an alternative to the constructivist approach to composition, i.e., the critical assembly of architectures of time according to some explicit scenario" (1991a, p. 31). By posing these techniques as an alternative, Beyls suggested that the two orientations are somehow incompatible. However, with the modular deployment of CA-generated values demonstrated here, it is clear that complex dynamical systems are just another technique, easily employed alongside conventional generative tools or direct temporal assembly.

Acknowledgments

Thanks to Paul Berg, Elizabeth Hoffman, William Kleinsasser, and the editors and anonymous reviewers of the *Journal* for valuable comments on earlier versions of this article. Thanks also to Jacques Chareyron and Peter Beyls for discussing their research, and to Pandelis Diamantides and Jan Trutzschler for research assistance.

References

Ariza, C. 2005a. "An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL." Ph.D. Dissertation, New York University.

Ariza, C. 2005b. "The Xenakis Sieve as Object: A New Model and a Complete Implementation." *Computer Music Journal* 29(2):40–60.

Ariza, C. 2006. "Beyond the Transition Matrix: A Language-Independent, String-Based Input Notation for Incomplete, Multiple-Order, Static Markov Transition Values." Unpublished manuscript, available online at www.flexatone.net/docs/btmimosmtv.pdf.

Banks, E. R. 1971. "Information Processing and Transmission in Cellular Automata." Ph.D. Thesis, Massachusetts Institute of Technology.

Battista, T., and M. Giri. 1988. "Composizione Tramite Automi Cellulari." In *Atti del VII Colloquio di Informatica Musicale.* Rome: Edizioni Arti Grafiche Ambrosini, pp. 181–182.

Beyls, P. 1989. "The Musical Universe of Cellular Automata." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 34–41.

Beyls, P. 1990a. "Musical Morphologies from Self-Organizing Systems." *Interface* 19(2–3):205–218.

Beyls, P. 1990b. "Subsymbolic Approaches to Musical Composition: A Behavioural Model." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 280–283.

Beyls, P. 1991a. "Chaos and Creativity: The Dynamic Systems Approach to Musical Composition." *Leonardo Music Journal* 1(1):31–36.

Beyls, P. 1991b. "Self-Organizing Control Structures Using Multiple Cellular Automata." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 254–257.

Beyls, P. 2003. "Selectionist Musical Automata: Integrating Explicit Instruction and Evolutionary Algorithms." *Brazilian Symposium on Computer Music,* n.p.

Beyls, P. 2004. "Cellular Automata Mapping Procedures." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 55–58.

Beyls, P. 2006. Personal correspondence, 16 September.

Bilotta, E., and P. Pantano. 2002. "Synthetic Harmonies: An Approach to Musical Semiosis by Means of Cellular Automata." *Leonardo Music Journal* 35(2):153–159.

Bowcott, P. 1989. "Cellular Automation as a Means of High-Level Compositional Control of Granular Synthesis." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 55–57.

Burdick, P., et al. 2004. "Music and NKS: Looking at and Listening to NKS." *NKS2004,* n.p.

Burraston, D. 2005. "Composition at the Edge of Chaos." *Proceedings of the 2005 Australasian Computer Music Conference,* n.p.

Burraston, D. 2006. "Generative Music and Cellular Automata." Ph.D. Thesis, Sydney University of Technology.

Burraston, D., and E. Edmonds. 2005. "Cellular Automata in Generative Electronic Music and Sonic Art: A Historical and Technical Review." *Digital Creativity* 16(3):165–185.

Burraston, D., et al. 2004. "Cellular Automata in MIDI-Based Computer Music." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 71–78.

Chareyron, J. 1988a. "Sound Synthesis and Processing by Means of Linear Cellular Automata." Unpublished poster presented at the International Computer Music Conference.
Chareyron, J. 1988b. "Wavetable come Automa Cellulare: una Nuova Tecnica di Sintesi." In *Atti del VII Colloquio di Informatica Musicale.* Rome: Edizioni Arti Grafiche Ambrosini, pp. 174–177.
Chareyron, J. 1990. "Digital Synthesis of Self-Modifying Waveforms by Means of Linear Automata." *Computer Music Journal* 14(4):25–41.
Collins, N. 2006. *Handmade Electronic Music: The Art of Hardware Hacking.* London: Routledge.
Comajuncosas, J. M. 1998. "Implementing Cellular Automata in Csound, with Some Musical Applications." Available online at www.csounds.com/jmc/Articles/cellular/synthesis.html.
Cook, M. 2004. "Universality in Elementary Cellular Automata." *Complex Systems* 15: 1–40.
Cowlishaw, M. 2005. "General Decimal Arithmetic Specification." Available online at: www2.hursley.ibm.com/decimal/decarith.html.
Dorin, A. 2002. "Liquiprism: Generating Polyrhythms with Cellular Automata." In *Proceedings of the 2002 International Conference on Auditory Display.* Kyoto: Advanced Telecommunications Research Institute, pp. 50–54.
Eldridge, A. C. 2002. "Adaptive Systems Music: Algorithmic Process as Musical Form." In *Proceedings of the 5th International Conference on Generative Art.* Milan: *Politecnico di Milano University.* Available online at www.generativeart.com/papersGA2002/13.pdf.
Elliott, J. M. G. 1998. "Isle Ex: Transmusic." Available online at http://jmge.net/tmusic.htm.
Ghazala, Q. R. 2004. "The Folk Music of Chance Electronics: Circuit-Bending the Modern Coconut." *Leonardo Music Journal* 14(1):97–104.
Hoffman, P. 1993. "Zelluläre Automaten als kompositorishce Modelle. Sind Chaos und Komplexität musikalische Phänomene?" In *Arbeitsprozesse in Physik und Musik.* Frankfurt am Main/Berlin: Peter Lang Verlag/Akademie der Künste Berlin, pp. 7–18.
Hoffman, P. 2002. "Towards an 'Automated Art': Algorithmic Processes in Xenakis' Compositions." *Contemporary Music Review* 21(2–3):121–131.
Hunt, A., and R. Kirk, R. Orton. 1991. "Musical Applications of a Cellular Automata Workstation." *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 165–168.
Karaca, I. 2005. "Random Precision: Some Applications of Fractals and Cellular Automata in Music Composition." Ph.D. Dissertation, The Ohio State University.
Karplus, K., and A. Strong. 1983. "Digital Synthesis of Plucked-String and Drum Timbres." *Computer Music Journal* 7(2):43–55.
Koenig, G. M. 1970. "Project Two: A Programme for Musical Composition." In *Electronic Music Report.* Utrecht: Institute of Sonology, p. 3.
Koenig, G. M. 1983. "Aesthetic Integration of Computer-Composed Scores." *Computer Music Journal* 7(4):27–32.
McAlpine, K., E. Miranda, and S. Hoggar. 1999. "Making Music with Algorithms: A Case-Study." *Computer Music Journal* 23(2):19–30.
Millen, D. 1990. "Cellular Automata Music." In *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 314–316.
Millen, D. 1992. "Generation of Formal Patterns for Music Composition by Means of Cellular Automata." In *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 398–399.
Millen, D. 2004. "An Interactive Cellular Automata Music Application in Cocoa." In *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 51–54.
Miranda, E. R. 1993. "Cellular Automata Music: An Interdisciplinary Project." *Interface* 22(1):3–21.
Miranda, E. R. 1995. "Granular Synthesis of Sounds by Means of a Cellular Automaton." *Leonardo* 28(4):297–300.
Miranda, E. R. 2003. "On the Music of Emergent Behavior: What Can Evolutionary Computation Bring to the Musician?" *Leonardo* 36(1):55–59.
Miranda, E. R., J. Correa, and J. Wright. 2000. "Categorising Complex Dynamic Sounds." *Organised Sound* 5(2):95–102.
Reiners, P. 2004. "Cellular Automata and Music: Using the Java Language for Algorithmic Music Composition." Available online at www-106.ibm.com/developerworks/java/library/j-camusic/?ca=dgr-lnxwj01j-camusic.
Ryan, J. 1991. "Some Remarks on Musical Instrument Design at STEIM." *Contemporary Music Review* 6(1):3–17.
Scaletti, C. 1987. "KYMA: An Object-Orientated Language for Musical Composition." In *Proceedings of the International Computer Music Conference.* San Francisco, California: International Computer Music Association, pp. 49–56.

Varga, B. A. 1996. *Conversations with Iannis Xenakis.* London: Faber and Faber.
Vercoe, B. 1986. *CSOUND: A Manual for the Audio Processing System and Supporting Programs.* Cambridge, Massachusetts: MIT Media Lab.
von Neumann, J. 1966. *Theory of Self-Reproducing Automata.* Urbana: University of Illinois Press.
Wolfram, S. 1983. "Statistical Mechanics of Cellular Automata." *Reviews of Modern Physics* 55:601–644.
Wolfram, S. 1985. "Origins of Randomness in Physical Systems." *Physical Review of Letters* 55(5):449–452.
Wolfram, S. 2002. *A New Kind of Science.* Champaign, Illinois: Wolfram Media, Inc.
Wuensche, A. 1999. "Classifying Cellular Automata Automatically: Finding Gliders, Filtering, and Relating Space-Time Patterns, Attractor Basins, and the *Z* Parameter." *Complexity* 4(3):47–66.
Wuensche, A., and M. Lesser. 1992. *The Global Dynamics of Cellular Automata: An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata.* Reading, Massachusetts: Addison-Wesley.
Xenakis, I. 1990. "Sieves." *Perspectives of New Music* 28(1):58–78.
Xenakis, I. 1992. *Formalized Music: Thought and Mathematics in Music.* Indianapolis: Indiana University Press.