Using Shape Grammar to Derive Cellular Automata Rule Patterns

Thomas H. Speller, Jr.[a,*]
Daniel Whitney[a]
Edward Crawley[b]

[a] *Engineering Systems Division, Massachusetts Institute of Technology, Cambridge, MA 02139-4307*

[b] *Department of Aeronautics and Astronautics, Engineering Systems Division, Massachusetts Institute of Technology, Cambridge, MA 02139-4307*

This paper shows how shape grammar can be used to derive cellular automata (CA) rules. Searching the potentially astronomical space of CA rules for relevance to a particular context has frustrated the wider application of CA as powerful computing systems. An approach is offered using shape grammar to visually depict the desired conditional rules of a behavior or system architecture (a form-function) under investigation, followed by a transcription of these rules as patterns into CA. The combination of shape grammar for managing the input and CA for managing the output brings together the human intuitive approach (visualization of the abstract) with a computational system that can generate large design solution spaces in a tractable manner.

1. Introduction

Cellular automata (CA) offer a large potential for simulating complex system dynamics using parallel computational processes. Challenges exist, however, in the practical utilization of CA. To date, researchers and practitioners continue to express uncertainty as to how to apply CA to modeling and simulation because of the immense difficulty in finding a set of rules from an astronomically large rule space [1] that might generate legitimate system architectures. Others have attempted to address finding CA rules by trial and error, genetic algorithms [2, 3], and genetic programming [4]. If the rule space can be managed, with proper representation of the underlying physics as demonstrated in the lattice gas research [5, 6], then CA may prove their applicability for generative systems. A CA methodology in complex system

*Corresponding author. Electronic mail address: tspeller@mit.edu.

modeling would be attractive because CA provide the advantages of parallel processing for large data sets with minimal overhead [3] and local neighborhood interactions comparable to nature's processes. Additionally, the CA approach is algebraic and logical, in that it permits using symbolic variables and functional operations according to specified rules. This allows the opportunity of mapping a visually depicted form-function (system architecture) directly into the CA and to present the output in a visual-spatial format as a designer would do.

A shape grammar is a formal set of rules applied to shapes to generate a language of design that allows the visualization of the desired form and function of the rules. The research objective herein is to exploit the potential of CA by presenting an approach for using shape grammar [7] to derive CA rules. The essential scope of this investigation falls within the domain of self-generative system architecture, with specific methodology entailing the use of a shape grammar for establishing the rules of the design process (at both the elemental and organizational levels). This is followed by a CA approach for actually generating the creative design space. The shape grammar thus expresses the material form relationships and the physics of the form-function and becomes transcribed into CA rules and their conditional neighborhoods, with the CA rules generating the design space.

Using shape grammars offers the system architect the capability of assuring that design proceeds by rules that embody the relevant principles of physics. The architect can then select for design candidates that meet stability, robustness, aesthetics, cost, and other requirements, thereby managing an otherwise possibly explosive design space. At the same time, shape grammars allow the system architect to explore a diverse variety of design styles, providing opportunities for the emergence of unexpected or unpredictable higher-order components and modular structures with potential usefulness. The system architect's role is thus to create a design space of conceptualizations and select good system architectures for a given specification, to determine the design physics and selection rules to be implemented, to develop the shape grammars to reflect these rules at the modular and hierarchical levels, and to program the CA to capture these rules and output a design catalog of the best candidates that meet the specification.

1.1 Cellular automata

CA comprise rules for evolving the state of a discrete dynamical system. The same rule is applied repeatedly to a system state, and new states are generated in parallel as a step function. As Wolfram [8] has emphasized, simple rules and programs can create complex systems. CA are local and parallel processors that can model physics in time and space. A rule, which can consist of a logical computation and/or pattern match

(such as to a list structure), is applied to each cell as based on the values of cells in its defined neighborhood, in order to determine its value at the next step. The origins of CA are attributable to von Neumann [9, 10], Ulam and Zuse [11], and Wiener [12], and its resurgence to Wolfram [13]. A survey of CA in the literature may be found in [1].

The simplest neighborhood is an elementary system consisting of a one-dimensional row of cells, each of which can contain the value 0 or 1 (depicted as two colors), with a local neighborhood of size 3 (range or radius of 1). More complex CA can be defined on two- or higher-dimensional arrays with multicolored cells and larger ranges. Each rule is represented as an array of cells. For the case of a local neighborhood of size 3, each triplet determines a single output cell in an array. A triplet with binary values can have eight possible patterns from 111 to 000. A local neighborhood of size 3 thus can generate 256 possible rules. The formula for calculating the rule size space in a one-dimensional system is $k^{k^{(2r+1)}}$, where k represents the color possibilities for each state and r is the range or radius of the neighborhood. It is interesting to note that merely increasing r from 1 to 2 and maintaining the colors at two increases the rule space from 256 to 4.3 billion.

Moreover, as illustrated in Figure 1, the number of parameter options under the discretionary control of the modeler (the "controllables") can produce a rule space that is beyond comprehension. Consequently, one of the fundamental difficulties in the application of CA is finding rules from this exploded space that exhibit the desired behavior.

CA can model spatio-temporal systems (e.g., biological, physical, or engineering) in which complex phenomena build up as a result of many simple local interactions. This local neighborhood seems similar to a "bounded rationality," as described by Herbert Simon [14], wherein

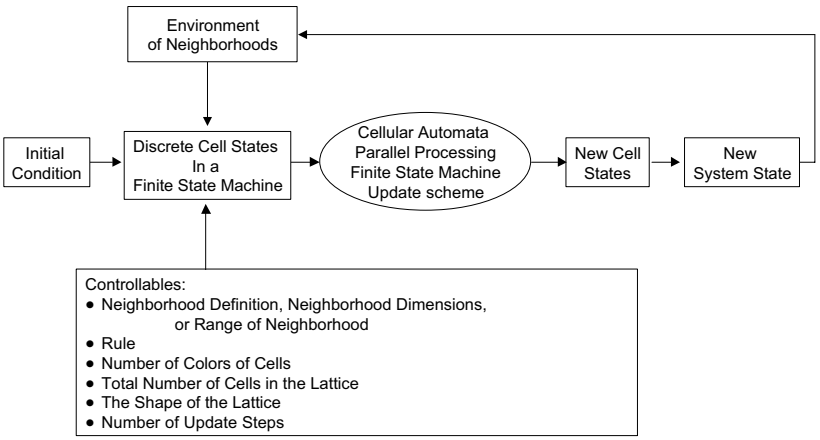


Figure 1. CA parameter space.

phenomena only act in local areas, do not explore all possibilities, and make a perfectly rational choice. The local neighborhood is a subset of full information, only a portion of perfect information is used for decision-selection at any moment as it is computationally intractable to consider full information in order to make a rational choice. The universe may be filled with such local neighborhoods which not only follow their own set of rules but also interact with other local neighborhoods in their universe, thereby collectively creating emergent behavior.

Lattice gas-fluid dynamics have been modeled successfully by CA rules using simple patterns of particle behavior adhering to the physical laws of conservation of momenta and conservation of mass [5, 15]. Most noteworthy is the use of a hexagonal neighborhood comprising 64 patterns of particle collision behavior which exactly predicts the Navier–Stokes equation of low Mach velocity gas dynamic behavior.

A different approach to finding a CA rule was taken by Hajela and Kim [3] in determining solutions to mechanical engineering problems. They used a genetic algorithm technique with crossover, mutation, and selection after testing for fitness to converge on a two-dimensional rule for finding the minimum energy of both a simple cantilever beam with a load applied at the end, and of a flat plate with a hole in its center and a pull force applied in all directions. Their paper goes on to cite two main difficulties with the current state of parallel processing, stemming from the more typical pattern of linear thinking by humans. First, existing programs are reworked ineffectively for use on different processors. Second, programs are written with different domains (as they put it) to be processed in parallel. They noted that there is a diminishing rate of return in processor speed as additional processors are added due to the necessary software overhead applied to control the different processors and the algorithm. CA, by nature being parallel, are therefore thought to be possibly a more natural way to conduct parallel processing.

1.2 Shape grammar

Shape grammar is a formal generative approach that has been applied to creating architectural forms. Its origins are attributable to the work of Stiny and Gips [7], reinforced by Mitchell [16, 17], Knight [18], and Cagan [19] (who has actually generated real product designs using shape grammars with customized output programs). Shape grammar is a precise and at the same time intuitive methodology in the visual medium for generating languages of design. Shape grammars can be used analytically, as in reverse engineering, for characterizing and classifying designs and patterns of designs, referred to as *styles* in architecture. A shape grammar includes a vocabulary of shapes and a set of spatial relations to control the positioning of shapes in the vocabulary. The shape rules are created and applied recursively starting with the initial

shape, and the generated designs compose a language. Operations and transformations may be applied to the shapes and the rules themselves. The practice of shape grammar in the field of architecture has focused primarily on form. However, functions and properties of shapes can be included in the grammar [20].

Formal grammars are systems of rules for characterizing the structure, or the syntax, of sentences in natural and artificial languages. Shape grammars are a geometrical design adaptation of Noam Chomsky's formal (phrase structure or transformational) grammars [21] and are recursively enumerable, having the capability of producing unrestricted languages [22]. Thus, shape grammars are systems of rules for characterizing the composition of designs in spatial languages. (See Figures 3 through 7 later for a demonstration of shape rules.)

As in Chomsky's formal grammars, which generate a language of one-dimensional strings, a shape grammar is also defined by a quadruple $SG = (V_T, V_M, R, I)$ but generates a language of two- or even three-dimensional objects that result in an assemblage of terminal shapes, where

V_T	is a set of terminal shapes (i.e., symbols)
V_M	is a set of markers (i.e., variables)
R	is a set of shape rules (addition/subtraction and euclidean transformations), $u \rightarrow v$ is the shape rule (i.e., productions; a production set of rules specifies the sequence of shape rules used to transform an initial shape to a final state and thus constitutes the heart of the grammar [23])
u	is in $(V_M \cup V_T)^+$ and v is in $(V_M \cup V_T)^*$; where + and $*$ refer to excluding or including the empty set
I	is the initial shape to which the first rule is applied (i.e., the start variable) [22].

A shape grammar applies rules to a given shape such as a rectangle by the accompanying operators, which may consist of shape addition, subtraction, and deletion. In addition, transformations are formally defined operations that specify how the components of grammars are modified to form new grammars. Transformations change the rules through the operators of translation, rotation, reflection, and scaling.

1.3 Other self-generating computational approaches

There is a considerable body of research on other approaches to bottom up self-generating design. Evolutionary algorithms arose from attempts to model the processes of nature as the ultimate designer, while tapping computers and programs to manage the time and space problems that confront man as the designer. Various evolutionary computation (EC) approaches have been developed according to darwinian processes of

evolution and have been applied in such endeavors as truss bridges [24], a sports car body and boat hull [25], LEGO® bridges [26, 27], tables [28, 29], circuits [4], programs, and others.[1] EC consists of at least four subapproaches: genetic algorithms (GAs) [30, 31], evolutionary programming (EP) [32], evolutionary strategies (ES) [33], and genetic programming (GP) [34]. EC approaches are characterized as having the following in common.

1. A given and usually random population of points (potential solutions) in the search for a solution to the fitness function.
2. Direct "fitness" information instead of function derivatives.
3. Evolutionary processes using probabilistic rather than deterministic transition rules (start with an initial population and the operators of selection, crossover, and mutation).
4. Evolution of solutions with parallel search for a solution to the fitness function.
5. Selection based on survival of the fittest.

Thus in the EC process, crossing genomic modular segments generates higher-order modules and greatly expands the design space (diversity). The principle of survival of the fittest leads to pruning out those designs with low or no probability of survival. However, ECs are a probabilistic set of methods that operate without regard to the laws of physics, discard possibly superior fits by not exploring the full combinatoric space, and are computationally bounded arbitrarily—halting is user defined [35]. This can lead to the generation of extraneous forms, which wastes computing time. Furthermore, there is no agreed upon method for deciding upon crossover and mutation percentages or location points. The more common GA approach also requires considerable hand of man (separate algorithmic fitness functions or tests) to physically examine the legitimacy of each structure created. The GA process is still at the beginning stage of being adopted for practical applications in real-world designs. Controlling the methodology seems to be the major issue in the use of GAs, but they are capable of generating designs quite rapidly and have been appealing for study across a variety of academic research purposes for over 30 years.

Lindenmayer systems (L-systems) are another kind of rule-driven design generation approach which entail the rewriting or substitution of strings of abstract symbols using production rules [36–38]. L-systems have been successful at generating designs emulating plant nature as

[1] For many examples refer to papers and proceedings from the Special Interest Group for Genetic and Evolutionary Computation of the ACM available at www.sigevo.org.

well as artificial architectural designs. Starting with an initial simple condition (axiom), production rewrite rules are recursively applied and can be geometrically interpreted. Typically, the rules are serially applied by an agent in order to generate complex forms [39]. L-systems differ from shape grammar in that their production rules operate on strings of symbols, whereas shape grammar rules act on shapes directly. Stauffer and Sipper [40] utilized an L-system as a one-dimensional character string generator, transforming the abstract output into a CA for a two-dimensional image interpretation. Noting some of the same problems with evolutionary design approaches, they felt that bridging L-systems and CA could offer a promising alternative to the study of self-replication and growth. The critical issues in this methodology, however, appear to be the extent of L-system rule design carried out by hand, especially considering its abstract form, and the ease of transformation to a viable CA format. Otherwise, there are interesting parallels between this combined approach and the shape grammar–CA approach proposed herein.

1.4 Comparison of shape grammar and cellular automata

A comparative examination of the shape grammar and CA methodologies reveals surprising complementarity (see Table 1). Essentially, both the shape grammar and CA approaches seek to identify whether a particular (local) pattern (shape or cell grid) is present and if so, provide rules to transform the recognized pattern as indicated. Shape grammars contain a set of rules applied to an initial condition as sequentially determined by the designer. While the CA approach typically runs out a single rule recursively, the designer/programmer certainly still has the power to selectively apply a series of CA rules by concatenating them into a production set order similar to a shape grammar. Since the local neighborhoods for CA, in copying real-life development, can capture both the function (what a system does, its purpose) and the form (the objects that deliver the function), there is no reason why a shape grammar cannot similarly do so through thoughtful rule definition. Thus, while shape grammars have tended to give results that were simple to perceive and conceive of but were too manually laborious, and CA have been easy to use for design generation but too abstract and unrestrained productively, both approaches serve as clear models for bottom up system architecting, utilizing basic elements and rules for transforming them as based on their patterns in a given space (context or neighborhood).

In summarizing the disadvantages and advantages of each computing methodology for generating system architectures, as mentioned earlier, a major problem with the CA approach has been that discovering self-generating rules for a particular intent out of an almost infinite

Basis of comparison	Shape grammar approach	CA approach
Handling of system architecture (form-function)	Form (shape) emphasis currently; function could be included	Has capability to have form and function encoded
Spatial design representation	A design is a set of shapes generated by rules starting with an initial state	A design is represented by sets of binary (or multicolor) cells on one-, two-, or higher-dimensional arrays
Design development	Provides ease of design and visualization	Ease of transcription for computation of the design
Intuitive use	Visually intuitive rule development for form and function	More difficult to visualize the final form-function to the neighborhood
Interpreter or compiler	Generative rules developed and sequenced by hand and applied by hand or machine	*Mathematica*® [41], the chosen programming language for this investigation, will run all rules automatically and manage the combinatorics and graphical output
Computing process	Recursive generation produces modules, hierarchies	Same
Pattern analyzing capability	Can be used to analyze design styles manually	Can be used easily as a pattern recognizer and may be suitable for examining cause–effect by reverse engineering

Table 1. Complementarity of shape grammar and CA.

CA rule space is beyond human capability. Furthermore, formulating design rules directly into CA format is an abstract and computationally complex task. On the other hand, as Professor Stiny[2] points out, there presently is no "robust" compiler or interpreter for shape grammars. This deficiency has limited the practical use of shape grammar to hand manipulation, which serves an educational value in itself, and by self-development of customized software which may have limited generalizability for other applications.

[2]Personal communication.

Despite these shortcomings, each approach has its advantages. The benefit of shape grammar lies in its ability to serve as a means of visual and intuitive coding that is highly adaptive to the visual and pattern recognition capability of the human brain. Shape grammar provides an intuitive feel in a concrete medium and can visually as well as conceptually represent physical phenomena. A production set can be developed in a logical manner that is easy to change or correct. Since CA are algebraic and logical, they can be easily programmed once a rule(s) is identified (such as by means of the language *Mathematica* [41]) for generating output from an initial condition. Large design spaces can be rapidly produced. The complementarity of shape grammar with CA therefore suggests that there is an opportunity to combine their strengths, offsetting their weaknesses, in order to derive CA rules for generating system architectures, or languages of designs, that will meet given specifications.

The principal research goal of this investigation, then, is to make a connection between shape grammar and CA that could lead to more effective modeling and design of legitimate system behaviors, with quick, extensive outputting of a solution space (system architectures that satisfy a specification). More specifically, the form-function captured in the shape grammar would be transcribed into the more abstract CA, mapping the physical relationships among shapes into a local neighborhood of cells to obtain the associated CA rules, which then provide the means for computation. This combination solves the problems of shape grammars being labor intensive for outputting the design space and CA being labor intensive for inputting data and rules.

2. The generalized shape grammar to cellular automata mapping approach

In order to address the stated research goal, this study demonstrates three stages for deriving and applying CA rules to develop solutions for a given problem. The shape grammar design methodology is applied first, followed by a shape grammar to CA (SG→CA) transcription step, and then the actual CA computational generation of solutions. Two examples, a simple LEGO bridge and the existing lattice gas model, are used to clarify this approach at an elementary level as research on this SG→CA approach is in its early development.

The first stage creates a shape grammar to visually depict the given problem (or specification) through a sequenced set of shape rules that are applied recursively to a defined position, beginning with an initial shape and ending with the specified goal. This stage entails decomposing the problem into its basic functional requirements, selecting the appropriate physical elements, modeling the physical laws applicable, and pictorially representing the physics as shape rules for the behavior of the elements in

relation to each other. Empirical investigation with the initial elements to understand how they combine together and how they fail provides necessary information for proceeding to the development of the shape grammar to address the problem at hand.

In the first example of the bridge, a single block shape was selected as the primitive, and experiments were conducted by hand with the calculation of mechanical statics to determine the minimum, or least action [42–45], modular shape configuration required to construct a column supporting a top row of blocks. The block was changed to a LEGO brick due to its additional connective force, which effectively expanded the diversity of columnar shapes and allowed for emerging interconnectivity. The second example of the lattice gas used a single particle as the primitive. Previous researchers [6] had discovered by iterative experimentation that 0 to 6 particles represented in a hexagonal star graph properly matched the Navier–Stokes equations. The interactive behavior of these particles was represented in this study as shapes in the form of picture graphs depicting states at time t and then state changes at time $t + 1$.

In the second stage, the shape rules are directly transcribed into CA rules following the logical definition of the neighborhood of shapes and their interrelationships as captured in the shape grammar. In actuality, the shape rules are transcribed into list mappings, which is an accepted alternative practice for defining CA rules, in order to simplify the construction of a neighborhood. The CA list rule structure is the neighborhood.

The brick shapes for the bridge were transcribed to numbers (or different colors), which were then expressed as neighborhood list mappings. Concatenating these list mappings as if assembling the bricks created the production set of rules for the basic bridge module. The lattice gas picture graphs were transcribed into a Moore neighborhood [46, 47] hexagonal format [11], then expressed as list mappings.

Once the CA rules are concatenated in a step comparable to sequencing the rules in a shape grammar, system solutions can be generated in the third stage. The CA rules are now read as a production set for subsequent graphical output.

After the production set constructed the basic bridge module, a combinatoric approach was employed to produce all the varieties of primary-order building modules and to combine them into higher-order modules. This differs from the lattice gas rule set which behaves similar to a multiagent system. The state at time t in a neighborhood is a list structure which is correspondingly mapped to another list pattern for time step $t + 1$. The lattice gas model can be initialized in any state, the number of evolutionary steps are selected, and the system execution proceeds in time steps, representing the behavior for the particle system while providing a complete history of state changes. Conservation of mass and momentum are preserved throughout the execution.

The SG→CA methodology is illustrated first with the development of a brick bridge using shape grammar as the design tool, followed by a transcription into CA rules for the computation of the designs. Then, by diagramming the rules determined by the lattice gas studies, a SG→CA application for the particle dynamics is demonstrated.

3. Case study of a bridge design

A LEGO bridge was used to investigate the properties and capabilities of the SG→CA method and to illustrate the results. This method succeeds in generating rules that build plausible bridges from elementary structural units that obey basic laws of static equilibrium, where the sum of all the forces and moments in the system equal zero.

3.1 Stage 1: Modeling of a five-row bridge using shape grammar methodology

An unrestricted shape grammar becomes helpful as a visual design aid for analyzing and synthesizing the elements and the rules that relate them for a given specification. The relevant physical constraints can be captured in a shape grammar as in the following procedure.

A shape grammar for a bridge begins with the selection of a building primitive, in this case a rectangular block or brick, and identification of the essential form-function concepts of a horizontal bridge deck and vertical column supports. This decomposition leads to the most basic modular assembly, a "T" structure composed of two blocks laid horizontally, supported by a single block. Figure 2 depicts a free body diagram of three blocks used to construct this T according to the principle of least action, showing the direction of the resultant force from each block's weight at its center of gravity. The arrows indicate the tipping moments of the blocks. If the primitive were a simple, smooth block, the neutrally stable support condition would require only a 50% block overlap to provide static equilibrium to the upper blocks. Weight alone creates the stability for these overlapped blocks.

Simple blocks positioned with 50% overlap is an ideal state requiring perfect balancing to prevent collapse, and a single block's fall can cause a cascading failure among its neighboring blocks. Consequently, there is a

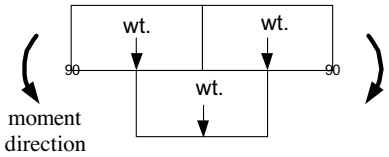


Figure 2. Basic assembly T module.

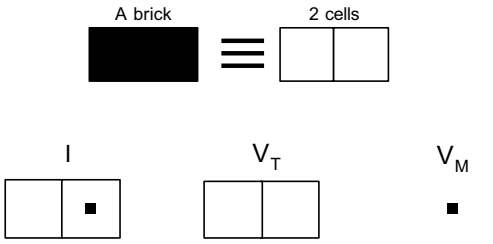


Figure 3. Shape grammar notation. I is the initial condition, V_T is a terminal shape with the markers erased, and V_M is a variable spatial marker for placement.

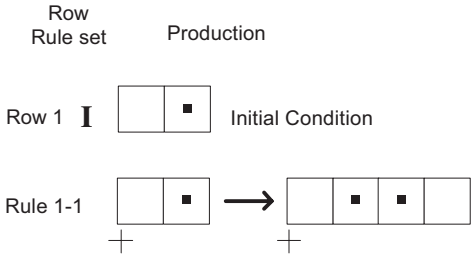


Figure 4. Shape rule depiction of bridge Row 1.

limitation to the number of stable block bridge architectures that can be designed. Unstable block columns do not become useful under further development as they cannot interconnect with other block columns to become stable. LEGO brick columns, on the other hand, use snap together joints that result in additional reacting moments beyond those created by weight alone. Providing such a connective force increases the possibility space for creative diversity due to the emergence of new structures with greater stability when LEGO columns are combined.

The basic assembly T module,[3] developed from an initial primitive Rectangular Brick, is defined by shape rules which embody this first design production. Per shape grammar notation (see Figure 3), the marker is a spatial label that indicates the proper juxtapositioning of the next added brick. These markers are later erased by terminal rules. The initial brick shape is represented by two squares (two cells in the CA lattice) for ease of SG→CA transcription.

Figure 4 shows an initial condition and a single rule that are then sequenced in a production rule set to build Row 1. The large boldface

[3]A *module* is an interconnected group of quasi-independent parts with functionality that can be reused in composing the system. It is believed that modularity is an integral property of hierarchical construction.

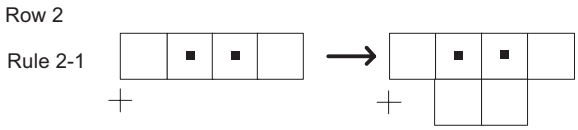


Figure 5. Shape rule depiction of the basic T module.

arrow is a mapping symbol meaning that the shape on the left-hand side is replaced by the shape on the right-hand side of the arrow. The + symbol indicates a reference "cross hair" for locating the rule output shape with respect to the input shape.

Row 1 is now the initial condition and an additional rule is applied to produce Row 2 to create the basic T module, which constitutes a section of the deck of the bridge, supported by the minimum number of bricks below, as is depicted in Figure 5.

The remaining rows supporting the basic T module are generated according to three possible equilibrium formations by which to add each next row's supports, the stable 50% offset of the brick to the left or 50% offset to the right (least action principle), in addition to keeping the brick straight inline for added diversity (see Figure 6).

For example, as shown in Figure 7, the five-row column module is created by concatenating the Row 1 and Row 2 rule sets, I,1-1,2-1 as defined in Figures 4 and 5, then applying a selected combinatoric from the L, R, and S rule sets (three concatenated rule sets for three rows), such as {L1,L2,R1,R2,S1,S2,L3,R3,S3} to produce the complete column.

3.2 Stage 2: Transcription of the five-row bridge design space into cellular automata

Using the shape grammar rule set to first define the cell characteristics and produce the T module, which will be combined with another module to make a higher-order bridge span module, the same rule set can then be transcribed easily as a CA for the purpose of managing the combinatoric task of generating a larger design space.

In Figure 8, an empty space (not occupied by a brick) in the shape grammar production is represented by a 0 (or the color white) in the corresponding cell within the CA lattice. The 1 (or color black) in a cell represents half of a brick, and the 2 (or third color) represents the ■ or spatial marker required to position the other half of the brick where needed. The actual CA rules for bridge Rows 1 and 2 are shown in Figure 9 in a one-dimensional CA neighborhood of size 3. In this example, the CA has six triplets as lists representing neighborhood rule mappings, which determine the next system state.

CA rules to capture the L, R, and S shape grammar options for generating additional support rows for the bridge are shown in Figure 10.

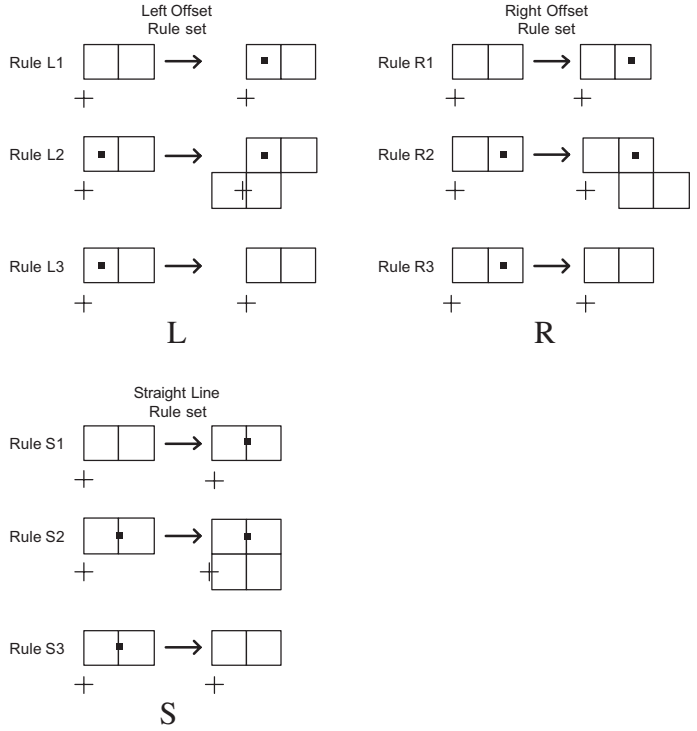


Figure 6. Shape rules for generating Rows 3 through 5 of the column.

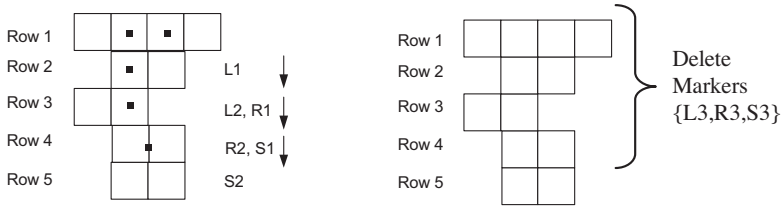


Figure 7. Five-row column module.

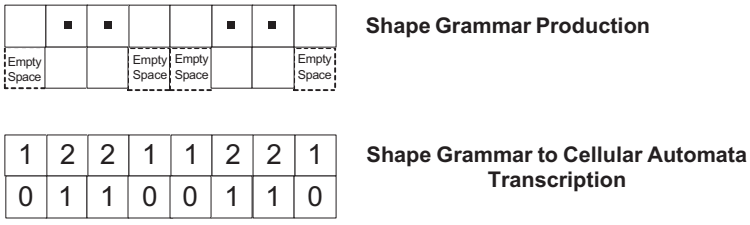
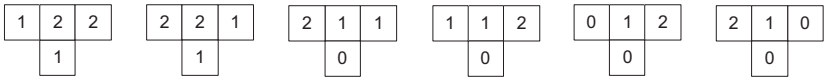


Figure 8. Transcribing the shape grammar of Rows 1 and 2 into a CA.



Neighborhood Rule Mapping using Lists:

$\{1,2,2\} \to \{1\}$ $\{2,2,1\} \to \{1\}$ $\{2,1,1\} \to \{0\}$ $\{1,1,2\} \to \{0\}$ $\{0,1,2\} \to \{0\}$ $\{2,1,0\} \to \{0\}$

Figure 9. CA representation of the shape grammar for Rows 1 and 2.

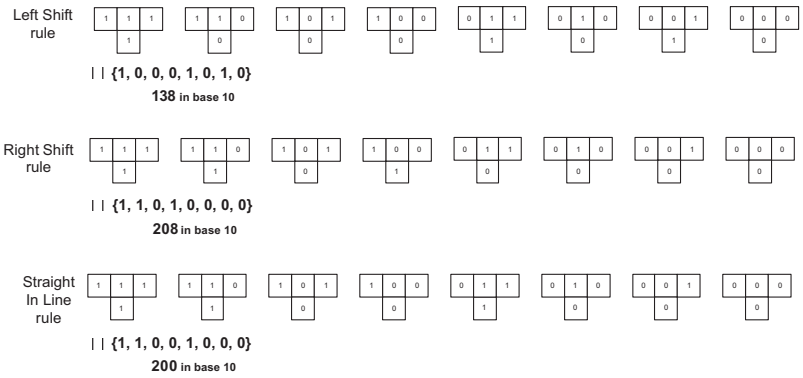


Figure 10. CA rule representation for Rows 3 through 5.

3.3 Stage 3: Generating system architectures for bridges

By concatenating the CA rules shown in Figure 10 combinatorically, a complete generation of all possible five-row column modules results in 27 designs as enumerated in Figure 11. These CA-generated column modules will be used as the basis for bridge supports in all further design/build combinations.

These columns are combinatorically paired into 729 higher-order modules as illustrated in Figure 12. The red (or gray) color denotes the redundant overlap of whole and half bricks, having the emergent effect of reducing the number of bricks in the structure.

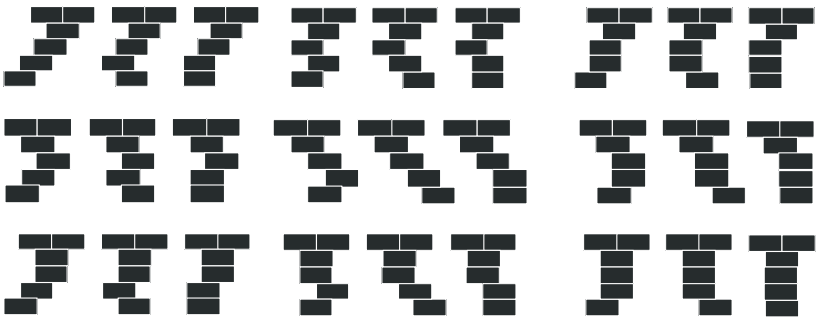


Figure 11. 27 column modules.

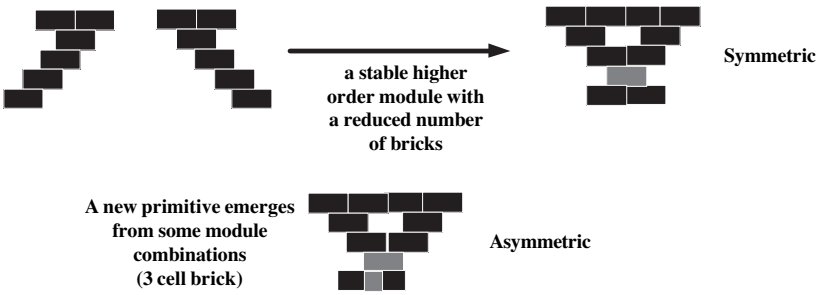


Figure 12. Emergence of diversity, new components, greater stability.

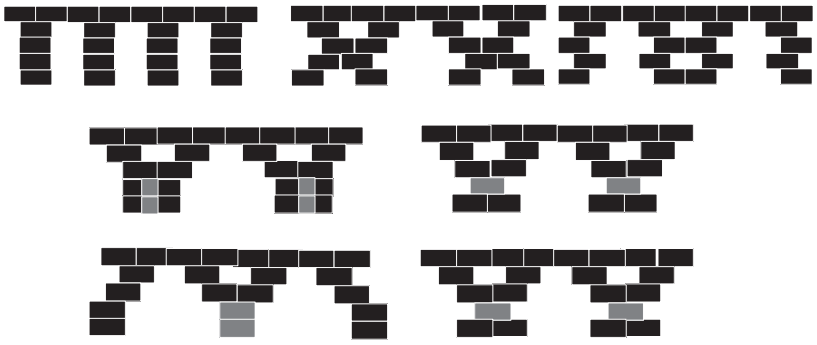


Figure 13. Sample of five-row bridge designs.

These higher-order modules can be further combined, such as by reflection or repetition, to create 1458 designs of a single-level bridge, with samples presented in Figure 13. Layering and scaling can extend the flexibility of options for creating bridges of greater dimension. Thus, the LEGO bridge example begins with just a single building primitive developed into a basic bridge component, to which three simple rules are applied recursively and combinatorically to create design complexity and diversity. Varying the pattern of reuse for this set of basic modules (by reflection or repetition) to complete the bridge greatly expands the possible solution space for bridge designs.

4. Example of shape grammar applied to the lattice gas problem

Because its neighborhood conditional rules can be enumerated visually for simple simulation, a lattice gas system lends itself to modeling by the SG→CA approach. Previously, the physical properties of thermodynamic and hydrodynamic systems had been captured by the discrete lattice gas CA models [48] exactly in accordance with the Navier–Stokes equations (referred to as FHP [6], which are the initials of the scientists'

last names). The lattice gas CA emulate the motion of molecules as deterministic and reversible. However, the discovery of the proper neighborhood for depicting the physical properties of fluids was not a trivial matter [5, 15]. Utilizing shape grammar first to represent the particle motion rules and then transcribing this shape grammar gas model into CA rules may have offered a more intuitive and therefore quicker route to the system architecture solution.

4.1 Stage 1: Modeling a lattice gas system using a shape grammar methodology

The relevance of lattice gas to this paper is that these researchers discovered and hand drew the enumerated particle collision patterns to fit within a CA neighborhood, which could have equivalently been expressed by a shape grammar with the embedded fluid dynamics. The physics of the lattice gas concept are well explained in the references provided herein. Briefly, the earlier HHP [49–51] model (HHP the initials of the scientists' last names) used a regular square neighborhood pattern that did not include the isotropy principle, resulting in approximate but not precise simulations of fluid motion. This prior trial for modeling lattice gas provides evidence that a shape grammar would only be as good as the researcher's ability to properly capture these behaviors in the production rules. The FHP model utilized a star graph with six directed edges to compose a production set of 64 possible rules (patterns). These graphs are compatible to a shape grammar approach of pictorial rule formulation. Collisions, if any, are arranged in a hexagonal pattern of edges at 60° to satisfy the isotropy as well as other required physical properties. While enlarging the neighborhood might seem to better capture the physics (but would make the model more complex), there is no appreciable gain in behavioral accuracy. Of note, head-on collisions could have an equal and opposite direction of ricochet, but the practice by lattice gas researchers has been to have the particles fly apart at opposite angled directions rotated left 60° in order to introduce random behavior [6].

The drawings in Figure 14 characterize the rules for this lattice gas language in the shape grammar format. They consist of patterns directly used to evolve the CA. The CA rules are approximations of the Navier–Stokes partial differential equations; however, these equations can only be solved for very simple contexts whereas the CA can model complex nonlinear behavior [48]. The convention incorporated for the particle flow is that arrows of orientation on the left-hand side (lhs) are directed inward to the neighborhood and on the right-hand side (rhs) are directed outward from the neighborhood. The FHP patterns start with the empty condition, the pattern (condition) of zero particles present in the neighborhood, and include the patterns for all possible particle movements with one to six particles present.

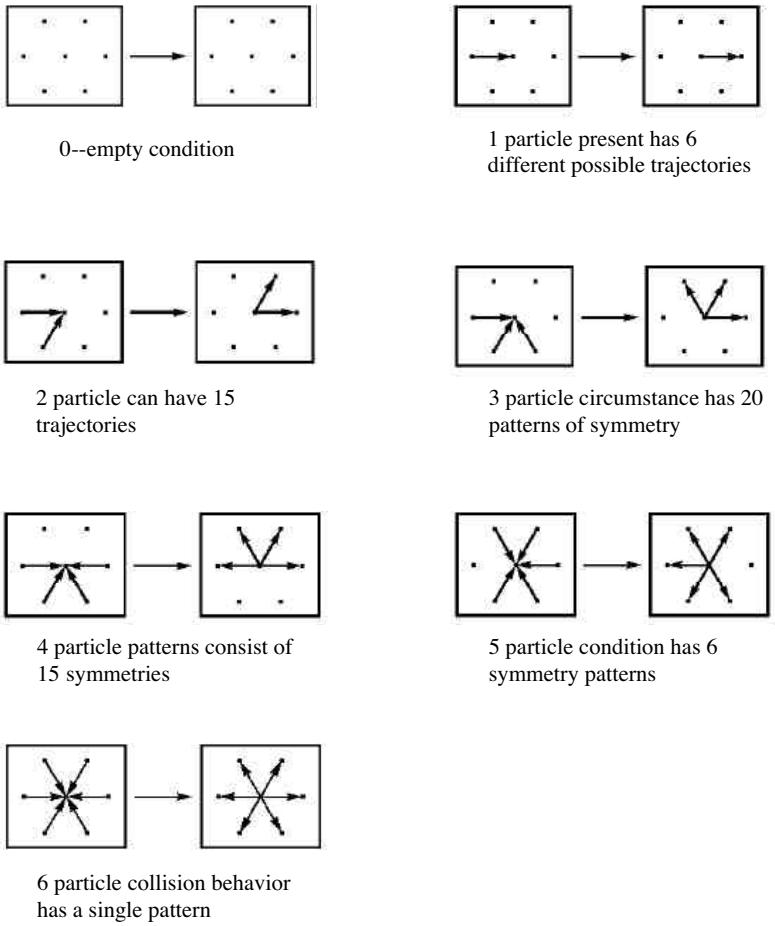


Figure 14. Example particle trajectories from each symmetry rule group.

4.2 Stage 2: Transcription of the lattice gas system design into cellular automata

These patterns of incoming and outgoing scattered particles in essence constitute shape rules in a shape grammar that captures the thermodynamic and fluid physical properties required to yield the proper behavior of fluid-like molecules. The SG→CA transcription can then proceed by the patterns themselves. Given six possible directed edge paths for particle motion, as indicated by the vertices of the hexagonal star graph in Figure 15 and reformulated into a nine-vertex star graph to accommodate a nine-cell CA neighborhood, there are $2^6 = 64$ possible particle flow patterns. Using the nine-vertex star format, the position of particles in these patterns can now be coded in binary form in the more typical

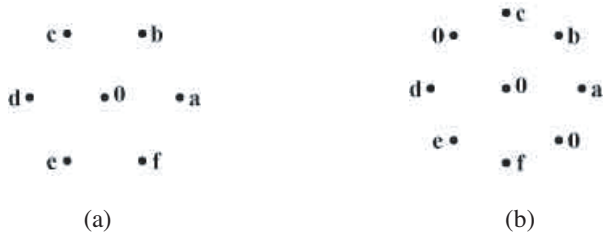


Figure 15. (a) Hexagonal star graph (seven vertices) and (b) nine-vertex star graph (0s showing the empty center and two unnecessary positions).

0	c	b
d	0	a
e	f	0

Figure 16. Nine-cell CA neighborhood matrix capturing the hexagonal format.

nine-cell CA (Moore) neighborhood, as shown in Figure 16. The state of the neighborhood at the beginning of the step function is the incoming condition of particles expressed as a six-element binary list. The letters read counterclockwise can contain a 1 or 0. The 0 cells have no effect on the neighborhood because those positions lack a particle or are not located within the hexagonal neighborhood of particle interaction.

The 64 collision patterns can be exhaustively enumerated as lists. Explicit replacement rules lhs→rhs contain the particle patterns. Figure 17 shows how the five-particle hexagonal pattern, which can be represented as this list,

$$\{1, 1, 0, 1, 1, 1\} \rightarrow \{1, 1, 1, 1, 1, 0\} \tag{1}$$

is equivalently expressed in the two-dimensional nine-neighborhood matrix. The other 63 particle collision patterns can be depicted in the same manner. The shape is converted to equivalent numeric symbols for computing since there is no interpreter or compiler for dealing directly with shapes as symbols. The CA rules are executed in parallel on a grid wherein the particles move according to their correct physical properties and conform to the Navier–Stokes equations for fluids and gases at subsonic velocities.

The HHP researchers seem to have based their original method for representing particles on the von Neumann neighborhood paradigm reduced to a square. However, their original neighborhoods produced particle patterns that were insufficient for capturing the physics of fluid dynamics. Subsequent research concentrated on finding a CA neighborhood that could more accurately represent the particle dynamics. Determining that the hexagonal particle pattern could be captured in

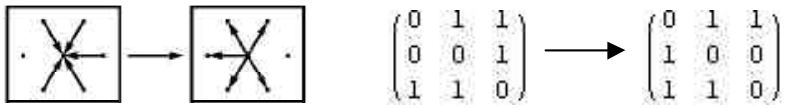


Figure 17. Five-particle shape rule and equivalent two-dimensional nine-neighborhood matrix.

a nine-cell Moore neighborhood, with the edges of the cells as particle movement lines, was a major leap from traditional CA paradigms. The physics may have been basically understood as evidenced by the reference to the Navier–Stokes equations, but the manner of incorporating the physics into an appropriate CA neighborhood, accounting for particle position and direction as well as empty spaces, was not easily foreseen. The lattice gas researchers thus had unknowingly set up a shape grammar for fluid particle behavior but had not been aware of the potential use of this pattern mapping approach for discovering the pertinent CA rules. (Stage 3 in this SG→CA, generating the system architecture for lattice gas, is described in FHP [6].)

In summary, the approach described in this section begins with the development of a lattice gas shape grammar based on an understanding of the physics of the particle dynamics. Diagrams representing the physics are used to create the actual shape rules that capture the necessary behaviors, without regard to finding an appropriate CA neighborhood. Of note, shape grammars are not restricted by the notion of neighborhood. The entire production rule set for emulating the physical behavior is converted to an algebraic topology in the form of binary lists derived straight from the fluid dynamics shape grammar interpreted in the format of an appropriately fitting neighborhood matrix. The lists represent patterns of physical behavior in the form of If–Then conditional statements, which effectively constitute a CA neighborhood. These patterns are essentially rules with the physics embedded and may be used now to compute a solution space *via* the CA encoding.

5. Conclusion

The purpose of this paper has been to show how shape grammars can be employed with cellular automata (CA) to more easily create a set of self-generative rules to achieve an intended system behavior. The notion of neighborhood in the CA framework serves as a conditional control on the behavior of the generating system. However, a shape grammar can also exhibit the essence of conditional neighborhood effects in that when a certain spatial arrangement among shapes is present, the rule condition can result in a developmental change to the neighborhood.

Thus, rules of nature or the physical world where context is a critical factor can be captured by both the shape grammar and CA approaches.

Trying all rule options for each step of development produces diversity in modular designs, which, when reused as nature does, results in a combinatoric explosion in the system's design space, a positive outcome as far as providing creative freedom. Within this very large design space may occur new properties or components (forms and functions) as a statistically likely outcome of the combinatorics and which normally cannot be humanly foreseen. Both shape grammar and CA, however to the negative, can yield unmanageable design spaces, and to constrain this result requires properly representing the underlying physics and laws of nature in the production set itself. This latter step is the basis for selection of the fittest and constitutes one of the primary responsibilities of the system architect.

Of importance, this shape grammar to CA (SG→CA) rule developing process might expand the use of shape grammar as a design input process with CA as the computational process for many self-generative systems, such as in biology, physics, and engineering. These tracks of possible future research may provide better understanding of the form and function development of system architectures (i.e., complex systems). Further applications of shape grammar for creating CA rules in order to satisfy specifications should be pursued to demonstrate the potential of this methodology as generalizable for diverse applications. Finally, this SG→CA procedure may permit researchers to more quickly simulate various conditions using different rules in order to observe their accuracy in conforming to actual nature, and also through trial and error replication allow researchers to find the correct physics for phenomena under investigation.

Acknowledgments

The authors express their appreciation to Dr. Una-May O'Reilly, Principal Research Scientist, MIT Computer Science and Artificial Intelligence Laboratory for her clarity improvement suggestions.

References

[1] N. Ganguly, *et al.*, "A Survey on Cellular Automata," Dresden University of Technology, Technical Report Centre for High Performance Computing, December, 2003.

[2] P. Hajela and B. Kim, "GA Based Learning in Cellular Automata Models for Structural Analysis," in *Third World Congress on Structural and Multidisciplinary Optimization*, Niagara Falls, NY, 1999.

[3] P. Hajela and B. Kim, "On the Use of Energy Minimization for CA Based Analysis in Elasticity," in *Proceedings of the 41st AIAA/ASME/ASCE/AHS SDM Meeting*, Atlanta, GA, 2000.

[4] J. Koza, *et al.*, *Genetic Programming III: Darwinian Invention and Problem Solving* (Morgan Kaufmann Publishers, San Francisco, 1999).

[5] D. H. Rothman and S. Zaleski, *Lattice-Gas Cellular Automata* (Cambridge University Press, Cambridge, U.K., 1997).

[6] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas Automata for the Navier–Stokes Equations," *Physical Review Letters*, **56** (1986) 1505–1508.

[7] G. Stiny and J. Gips, "Shape Grammars and the Generative Specification of Painting and Sculpture," in *Information Processing 71*, edited by C. V. Freiman, Amsterdam, 1972.

[8] S. Wolfram, *A New Kind of Science* (Wolfram Media, Inc., Champaign, IL, 2002).

[9] J. von Neumann and A. W. Burks, *Theory of Self-Reproducing Automata* (University of Illinois Press, Urbana, 1966).

[10] J. von Neumann, *The Computer and the Brain* (Yale University Press, New Haven, 1958).

[11] A. Ilachinski, *Cellular Automata, a Discrete Universe* (World Scientific, New Jersey, 2001).

[12] N. Wiener and A. Rosenblueth, "The Mathematical Formulation of the Problem of Conduction of Impulses in a Network of Connected Excitable Elements, Specifically in Cardiac Muscle," *Archivos del Instituto Cardioligia de Mexico*, (1946) 205–265.

[13] S. Wolfram, "Statistical Mechanics of Cellular Automata," *Review of Modern Physics*, **55** (1983) 601–644.

[14] H. A. Simon, "A Behavioral Model of Rational Choice," *Quarterly Journal of Economics*, **69** (1955) 99–118.

[15] T. Toffoli and N. Margolus, *Cellular Automata Machines* (The MIT Press, Cambridge, MA, 1987).

[16] W. J. Mitchell, "A Computational View of Design Creativity," in *Modeling Creativity and Knowledge-Base Creative Design*, edited by E. Gero and M. Maher (Lawrence Erlbaum Associates, Hillsdale, NJ, 1993).

[17] W. J. Mitchell, *The Logic of Architecture* (The MIT Press, Cambridge, MA, 1990).

[18] T. W. Knight, *Transformations in Design* (Cambridge University Press, Cambridge, U.K., 1994).

[19] J. Cagan and E. Antonsson, editors, *Formal Engineering Design Synthesis* (Cambridge University Press, Cambridge, U.K., 2001).

[20] G. Stiny, "Weights," *Environment and Planning B: Planning and Design*, **19** (1992) 413–430.

[21] N. Chomsky, *Syntactic Structures* (Mouton & Co., The Hague, 1957).

[22] G. Stiny, *Pictorial and Formal Aspects of Shape and Shape Grammars* (Birkhauser Verlag, Basel, 1975).

[23] P. Linz, *An Introduction to Formal Languages and Automata, Third Edition* (Jones & Bartlett Publishers, Sudbury, MA, 2001).

[24] K. Deb, Surendra Gulati, and Sekhar Chakrabarti, "Optimal Truss-Structure Design using Real-Coded Genetic Algorithms," in *Genetic Programming 1998: Proceedings of the Third Annual Conference* (University of Wisconsin, Madison: Morgan Kaufmann, 1998).

[25] P. J. Bentley, "Genetic Evolutionary Design of Solid Objects using a Genetic Algorithm," Dissertation, University of Huddersfield, 1996.

[26] P. Funes, "Evolution of Complexity in Real-World Domains," Dissertation, Brandeis University, 2001.

[27] P. Funes and J. Pollack, "Computer Evolution of Buildable Objects," in *Fourth European Conference on Artificial Life*, edited by P. H. a. I. Harvey (The MIT Press, Cambridge, MA, 1997).

[28] G. S. Hornsby, "Generative Representations for Evolutionary Design Automation," Dissertation, Brandeis University, 2003.

[29] G. S. Hornsby, H. Lipson, and J. B. Pollack, "Generative Representations for the Automated Design of Molecular Physical Robots," *IEEE Transactions on Robotics and Automation*, **19**(4) (2003) 703–719.

[30] J. Holland, "Genetic Algorithms and the Optimal Allocations of Trial," *SIAM Journal of Computing*, **2**(2) (1973) 88–105.

[31] J. Holland, *Adaptation in Natural and Artificial Systems* (The MIT Press, Cambridge, MA, 1992).

[32] P. J. Bentley, editor, *Evolutionary Design By Computers* (Morgan Kaufmann Publishers, San Francisco, 1999).

[33] L. Rechinberg, "Evolution Strategy," in *Computational Intelligence—Imitating Life*, edited by J. Zurada, R. Marks II, and C. Robinson (IEEE Press, Piscataway, NJ, 1994).

[34] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (The MIT Press, Cambridge, MA, 1992).

[35] J. Kennedy and R. C. Eberhart, *Swarm Intelligence* (Morgan Kaufmann Publishers, San Francisco, 2001).

[36] P. Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants* (Springer-Verlag, New York, 1990).

[37] P. Prusinkiewicz and James Hanan, "Lindenmayer Systems, Fractals, and Plants," *SCIAM*, **33**, 1991 (The Society for Industrial and Applied Mathematics).

[38] A. Lindenmayer, "Mathematical Models for Cellular Interactions in Development I: Filaments with One-sided Inputs," *Journal of Theoretical Biology*, **18**(3) (1968) 280–299.

[39] P. Testa, *et al*., "MoSS: Morphogenetic Surface Structure, A Software Tool for Design Exploration," in *Proceedings of Greenwich 2000 Digital Creativity Symposium* (Greenwich, 2000).

[40] A. Stauffer and M. Sipper, "On the Relationship between Cellular Automata and L-systems: The Self-replication Case," *Physica D: Nonlinear Phenomena*, **116**(1–2) (1998) 71–80.

[41] *Mathematica* (Wolfram Research, Inc., Champaign, IL, 2007).

[42] R. P. Feynman, *The Feynman Lectures on Physics, Volumes I–III* (Addison-Wesley, Boston, 1963). Specifically see "The Principle of Least Action" lecture in Volume II, 19-1.

[43] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics, Third Edition* (Addison-Wesley, Boston, 2002).

[44] G. J. Sussman, J. Wisdom, and M. E. Mayer, *Structure and Interpretation of Classical Mechanics* (The MIT Press, Cambridge, MA, 2001).

[45] E. F. Taylor, "Principle of Least Action," (2005); www.eftaylor.com/leastaction.html.

[46] E. F. Moore, "Machine Models of Self-reproduction," *American Mathematical Society Proceedings of Symposia in Applied Mathematics*, **14** (1962) 17–33.

[47] E. F. Moore, "Machine Models of Self-reproduction," in *Proceedings of the Symposium on Mathematical Problems in the Biological Sciences*, 1961.

[48] S. Wolfram, *Cellular Automata and Complexity, Collected Papers* (Addison-Wesley, Reading, MA, 1994).

[49] J. Hardy and Y. Pomeau, "Thermodynamics and Hydrodynamics for a Modelled Fluid," *Journal of Mathematical Physics*, **13** (1972) 1042.

[50] J. Hardy, Y. Pomeau, and O. de Pazzis, "Time Evolution of a Two-dimensional Model System," *Journal of Mathematical Physics*, **14** (1973) 1746–1759.

[51] J. Hardy, Y. Pomeau, and O. de Pazzis, "Molecular Dynamics of a Lattice Gas: Transport Properties and Time Correlation Functions," *Physical Review A*, **13** (1976) 1949–1961.