# Shape Annealing: A New Approach for Controlling Shape Generation

J. Cagan, W. Mitchell

EDRC 24-68-91

# Shape Annealing:
# A New Approach for Controlling Shape Generation

by

Jonathan Cagan
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
and
William J.Mitchell
Graduate School of Design
Harvard University
Cambridge, MA 02138

## 0. Abstract

The new design paradigm of *shape annealing* is introduced. Shape annealing is a variation of the simulated annealing stochastic optimization technique. It produces optimally directed shapes within the language specified by a shape grammar. It does so by controlling the selection and application of shape rules such that superior shapes evolve as the algorithm **progresses.**

## 1. Introduction

Shape grammars, as defined by Stiny (1980), have successfully been employed to generate designs for many different types of artifacts - villas in the style of Palladio (Stiny and Mitchell, 1978), Mughul gardens (Stiny and Mitchell, 1980), prairie houses in the style of Frank Lloyd Wright (Koning and Eizenberg, 1981), Greek meander patterns (Knight, 1986), and suburban Queen Anne Houses (Flemming, 1987), to name just a few. A shape grammar derives designs in the language which it specifies by recursive application of shape transformation rules to some starting shape. In general, at any step in the derivation of a design, there is a choice to be made among different possibilities for rule application. A particular design in the language results from a particular sequence of choices: it is derived by taking a distinctive path through the state-action tree of the grammar, from the starting shape to one of the terminal nodes.

The literature of shape grammars and their applications contains little discussion of *how* these choices among alternative rule applications might appropriately be made. Indeed, the issue is usually avoided, and grammars are routinely demonstrated by exhaustively enumerating possible derivations or by randomly sampling them. Here we propose a different approach — specifically, the use of optimization criteria to control the derivation of shapes. We formulate the design problem as an optimization problem with a quantifiable objective function and a set of design constraints. We then utilize a variation of the stochastic optimization technique known as simulated annealing, which we call *shape annealing,* to control choice among alternative rule applications as shapes are derived. The result is an *optimally directed* design solution.

Cagan and Agogino (1991) define optimally directed design as an approach to design which **attempts** to determine optimal regions of the design space by directing search toward improving **the** objectives and eliminating suboptimal or dominated regions. The result is not necessarily a globally optimal numerical solution, but rather insight into how the design might be improved relative to the objectives, and a superior -- that is, optimally directed -- design solution.

Simulated annealing procedures for production of such optimally directed designs generate random perturbations of a given solution state. They absolutely accept the new

solution **if it** is **better,** or accept it with a certain probability if it is worse. Past applications of **simulated** annealing have been to determine the optimal solution of continuous or **ordered discrete** design variables within a *fixed* design configuration: the form of the solution was known but specific values needed to be determined. Our *shape annealing* procedure, by contrast, uses a variation of simulated annealing not to determine the values of variables in a *given* design configuration, but rather to *generate* the configuration itself. Application of this procedure results in evolution of designs: the fittest (most optimal) configurations survive. It requires one or more design criteria (objective functions) with a set of design constraints, a starting shape, and a set of shape transformation rules.

The next section summarizes the principles of simulated annealing. Our shape annealing procedure is then described. Finally, shape annealing is demonstrated with a simple shape grammar which can generate a countably infinite set of different shapes.

## 2. Simulated Annealing

Simulated annealing is a stochastic optimization technique which has been demonstrated to solve continuous *(e.g.,* Jain, *et ai,* 1990; van Laarhoven and Aarts, 1987, Cagan and Kurfess, 1991) or ordered discrete *(e.g.,* Kirkpatrich, *et al*$_y$ 1983; Jain and Agogino, 1990) optimization problems of fixed structure. Kirkpatrich, *et al.*$_y$ (1983) developed the simulating annealing algorithm based on Metropolis[1] Monte-Carlo technique (Metropolis, *ex ai,* 1953). The idea is analogous to the annealing of metals. A cooling schedule is defined giving a temperature reduction over a certain number of iterations. Temperature is a gradient variable with no physicalmeaning; the variable is called *temperature* to maintain the analogy with metal annealing. At high temperatures selection of a solution point is quite random **while at** lower temperatures the solution is more stable; the metal annealing analogy is that **at high** temperatures the molecules are at a highly random state while at lower temperatures they reach a stable minimum energy state.

With simulated annealing, a feasible state, *si,* is randomly selected and the energy at that state, $E_{sl>}$ is evaluated. A different feasible state, *S2,* is then selected and evaluated to $E_{s2}$. If $E_{\$}2 > E_{S}j$, then *S2* becomes the new solution state. If $E_{J2} \leq$ £$/» ^enl^ere ⁄sa probability, based on the temperature, that the new state will be accepted anyhow.

Acceptance is determined by the probability calculation:

$$\Pr\{E_{s_2}\} = \frac{E}{Z},$$

where Z(T) is a normalization factor. A random number, r, between 0 and 1 is generated and compared with $\Pr\{E_{52}\}$. If $r < PT[E_{S2}]$ then the new state is accepted anyhow; otherwise the old state is retained. The temperature is reduced and the process continues until convergence is reached or the temperature reaches 0. Generally, the size of the mutation space is also reduced and asymptotes to zero. In this case, it can be proven (Lundy and Meese, 1986) that if *equilibrium (i.e.,* convergence) is reached at each temperature and if the temperature is reduced slowly enough, then the algorithm is guaranteed to determine the global optimum. Because sufficient time cannot be guaranteed for large problems, simulated annealing is used to search only for a good solution and the exact globally optimal solution is often sacrificed for computation time.

Generally, the algorithm is run for several iterations at a given temperature until equilibrium is reached or until a certain number of iterations has occurred. The temperature is then reduced by a fixed amount and the algorithm is again run until convergence or an iteration limit is achieved. When there is no accepted new solution at a given temperature the minimum has been found.

Simulated annealing has only been used to evaluate a solution for a fixed design configuration. In the next section we extend the algorithm to actually *generate* the design configuration based on optimization criteria.

## 3. Shape Annealing

In our shape annealing algorithm, simulated annealing is used to determine whether a randomly selected shape rule should be applied at a given design state. Given a current design state, an eligible rule is selected and applied to the design. If the new design does not violate any constraints then it is sent to the Metropolis algorithm which compares the

new state to the old state and, based on the temperature profile, it determines whether to accept it or not. If the rule violates a constraint then the old state is maintained as the current. Note that a rule may not be eligible for application. The current algorithm actually selects any possible rule and then checks to verify that the rule is eligible.

By only applying additive rules, the design may rapidly converge on an inferior solution because it can work itself into a state where no rule can be applied without a constraint violation. Our algorithm can back itself out of these local solutions by reversing the previous rule. For every additive rule, there exists a subtractive opposite; if the subtractive rule is fired immediately following its companion additive rule then the effect of the additive rule is removed.

The result of modeling a subtractive rule for every additive rule is an evolutionary design generation system. As the shape evolves, if it starts to converge on an inferior solution, the shape can backtrack and perturbate to a superior design configuration. This perturbation occurs from generating new design shapes and does not require maintaining a trail of all design decisions; the current state is all that is known about the design rather than all the mutations which had tried and failed during its evolution.

The shape annealing algorithm is given in Figure 1. The number of outer loop iterations (i.e., the determination of reduction_factor) and the the number of inner loop iterations, n, need to be determined for the specific problem based on convergence of good solutions. Note that the size of the mutation space is not reduced at each iteration; this deviates from the normal simulated annealing approach but is required because there is no metric from which to evaluate a given configuration relative to another except through the objective function. In actuality, the size of the design space *increases* while the algorithm runs; however, the increase is controlled and the algorithm finds a way to generate an optimally directed design from a countably infinite number of possible configurations.

Shape grammars may be able to generate an infinite number of potential solutions. Given design constraints the number may be constrained to a finite but huge number. It may be impossible for a human or computer to generate all possible shapes in a reasonable period of time. With the shape annealing algorithm, a good solution evolves in polynomial time. It is not guaranteed to be the global minimum; however, by letting the algorithm run

a sufficient time, our experience shows the solutions to be quite good. An example is presented in **die** next section.

## 4. Example

As an example application of the shape annealing algorithm, consider the shape grammar of half-hexagons shown in Figure 2 from Mitchell (1990). Given this language, a countably infinite number of shapes can be generated. As a designer, we desire to place as many half-hexagons as possible into a given space, without overlap, while satisfying the grammar of Figure 2. For purposes of illustration, we consider all spaces to be of a constant area (25 units) and the half hexagon has a long base of one unit and a short base of one half unit Rule 4 is given for completeness but is actually only applied after the final solution is found.

We pose this problem as an optimization problem as:

maximize: number of pieces
subject to: pieces are assembled based on shape grammar,
no pieces can overlap;
shape must fit in bounds of defined space.

For this discussion, we limit the defined space to be rectangular and implement the grammar based on the center line of the half-hexagon (thus the actual rectangle will be slightly larger than 25 square units).

For implementation, the design state is stored by a trace of rule numbers; thus the detailed design is not stored but only a string of rules which create the design is stored. A new rule **number** is added by storing it at the end of the string. Implementation of the rule reversal occurs by randomly generating both positive and negative rule numbers. If a negative rule number is generated and the previously applied rule is the analogous positive number, then the positive rule is removed from the rule* string.

Figures 3-6 show shapes generated with shape annealing for a square boundary space, and Figures 7-9 for rectangular spaces. The fill pattern indicates which rule was applied:

the light fill indicates rule 1, the medium fill indicates rule 2, and the dense fill indicates rule 3. Each Figure starts with a piece in the bottom left comer. Figures 5 and 6 are of particular note. As discussed in section 3, as a shape is being generated it can work its way into a poor solution which, without rule reversal, has no further improvement. Because of the rule reversal, the shape can work its way out of the inferior state and then progress toward a better, optimally directed design. Figures 5a and 6a demonstrate solution states generated on the way to creating the final states of Figures 5b and 6b, respectively. Note that in Figures 5a and 6a there are no other valid additive rules which can be applied from the end state; obviously these solutions are inferior designs. As the shape annealing algorithm progressed, the reversal rules became dominant and eventually the shape was able to work out of the "corner" and found a better route with which to progress.

## 5. Discussion

Examining the shapes of Figures 3-9, a few observations are relevant:

• The final configuration is quite sensitive to the initial path generated. At the very beginning of the annealing process, any shapes generated will be better, based on the objective function, than the previous states. The shape quickly goes off in the initial directions generated and has no reason to backtrack to improve the beginning part of the shape. Thus, if a smart initial path is not chosen, the algorithm may not be able to densely pack portions of the space.

• Initially there are a countably infinite number of shapes which can be generated; for 3 possible generative shapes at each state with a half-hexagon of .325 square units area and a 25 square units specified space, the worst case could be $3^{75}$ possible paths since up to 75 pieces could potentially fit in the space. Because there are constraints which prevent overlap and define bounds on the shape, and because the grammar is designed to prevent complete packing, this number is considerably reduced; however, the number of possible designs is combinatoric and quite large. Shape annealing is able to determine very good solutions in polynomial time.

• The **variance** of the shape at a given temperature is sensitive to the number of possible states which remain feasible. Because there are so many available shapes at the start, there is a great variance in the shape after each initial temperature step. However, after much of the shape is generated, it is harder to generate a shape which does not violate a constraint; thus the rate of change of the shape is smaller. The result is a shape which is more likely to be sparse near the start and dense near the end.

• The grammar of Figure 2 can be extended as shown in Figure 10. This results in much denser configurations (and thus higher objective functions). The more rules which are available and the less constraints on the design space, the more creative the configuration which can be generated.

• The algorithm is computationally efficient There is no need to maintain a trace of design decisions; only the current state and proposed mutation at each step need be stored. The algorithm itself will backtrack out of a configuration if required.

• We emphasize that the designs generated by shape annealing are not guaranteed to be globally optimal. Rather, the resulting designs are generally very good solutions to an otherwise intractable problem.

• The shapes generated by shape annealing *evolve* into their final configuration, stimulating an analogy to the natural process of evolution. Although mutations may be pursued even if initially they do not improve the design, the survival of only the fittest designs remain.

## 6. Conclusions

We have introduced a powerful method called *shape annealing* to control the generation of shapes with shape grammars based on optimally directed solutions to functional needs. The application of the shape annealing algorithm to a simple shape grammar has led to the evolution of a large variety of shapes which satisfy the given design requirements. Shape annealing introduces a new way of applying shape grammars to the design process.

## 7. Acknowledgements

The authors would like to thank John Corson-Rikert for providing the graphical interface to draw the figures in this paper. This work was partially sponsored by the Engineering Design Research Center at Carnegie Mellon University, an NSF sponsored research center.

## 8. References


Cagan, J. and A.M. Agogino (1991), "Inducing Constraint Activity in Innovative Design", in press: *AI EDAM: Artificial Intelligence in Engineering, Design, Analysis and Manufacturing,* 5(1).

Cagan, J., and T.R. Kurfess (1991), "Optimal Design for Tolerance and Manufacturing Allocation", *EDRC Report 24-67-91,* Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213.

Hemming, U. (1987), "More than the Sum of the Parts: the grammar of Queen Anne Houses", *Environment and Planning* £, 14:323-350.

Jain, P., and A.M. Agogino (1990), "Theory of Design: An Optimization Perspective", *Mech. Mack. Theory,* 25(3):287-303.

Jain, P., P. Fenyes, and R. Richter (1990), "Optimal Blank Nesting Using Simulated Annealing", *Proceedings of: ASME Design Automation Conference: Advances in Design Automation -1988* (Ravani, ed.), 2:109-116.

Kirkpatrick, S., C D. Gelatt, Jr., and M.P. Vecchi (1983), "Optimization by Simulated Annealing", *Science,* 220(4598):671-279.

Knight, T.W. (1986), "Transformations of the Meander Motif on Greek Geometric Pottery", *Design Computing,* 1:29-67.

Koning, H., and J. Eizenberg (1981), "The Language of the Prairie: Frank Lloyd Wright's Prairie Houses", *Environment and Planning* fl, 8:295-323.

Lundy, and Mees (1986), "Convergence of an Annealing algorithm", *Mathematical Programming,* 34:111-124.

Mitchell, W.J. (1990), *The Logic of Architecture,* MIT Press, Cambridge, MA, p.143.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller (1953), *J. Chem Phys.,* 21: 1087-1091.

Stiny, G. (1980), "Introduction to Shape and Shape Grammars", *Environment and Planning B,* 7:343-351.

Stiny, G., and WJ. Mitchell (1978), "The Palladian Grammar", *Environment and Planning B,* 5:5-18.

Stiny, G., and W.J. Mitchell (1980), "The Grammar of Paradise: on the Generation of Mughul Gardens", *Environment and Planning B,* 7:209-226.

van Laarhoven, PJ.M., and E.H.L.Aarts (1987), *Simulated Annealing: Theory and Applications,* D.Reidel Publishing Co.

```
Begin ANNEAL
    T - 1 .
    Define initial state;
    Evaluate state;
    While T>O do
        success = 0;
        For mutations = 1 to n do     /* at each temperature mutate n times
                                         until convergence or limit reached */
            Let temp_state = state;
            Generate rule;
            If rule is applicable
            Then Begin
                apply rule to temp_state;
                if verify constraints of temp_state
                Then Begin
                    Evaluate temp_state;
                    Test temp_state with Metropolis;
                    If acceptable
                    Then
                        state = temp_state;
                        success = success + 1;
                    End
                End
            End
            If success > limit then break;
        End
        If success = 0 then break;          /* no improvement...
                                               solution found */
        T = T*reduction_factor,
    End
End
```

**Figure 1** Shape Annealing Algorithm

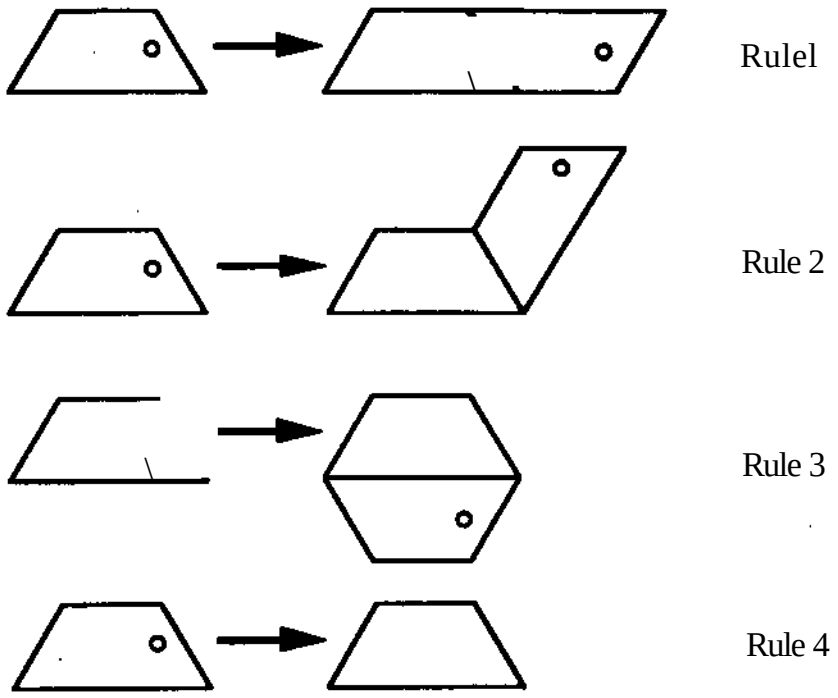



**Figure 2** Example Shape Grammar (from Mitchell, 1990)

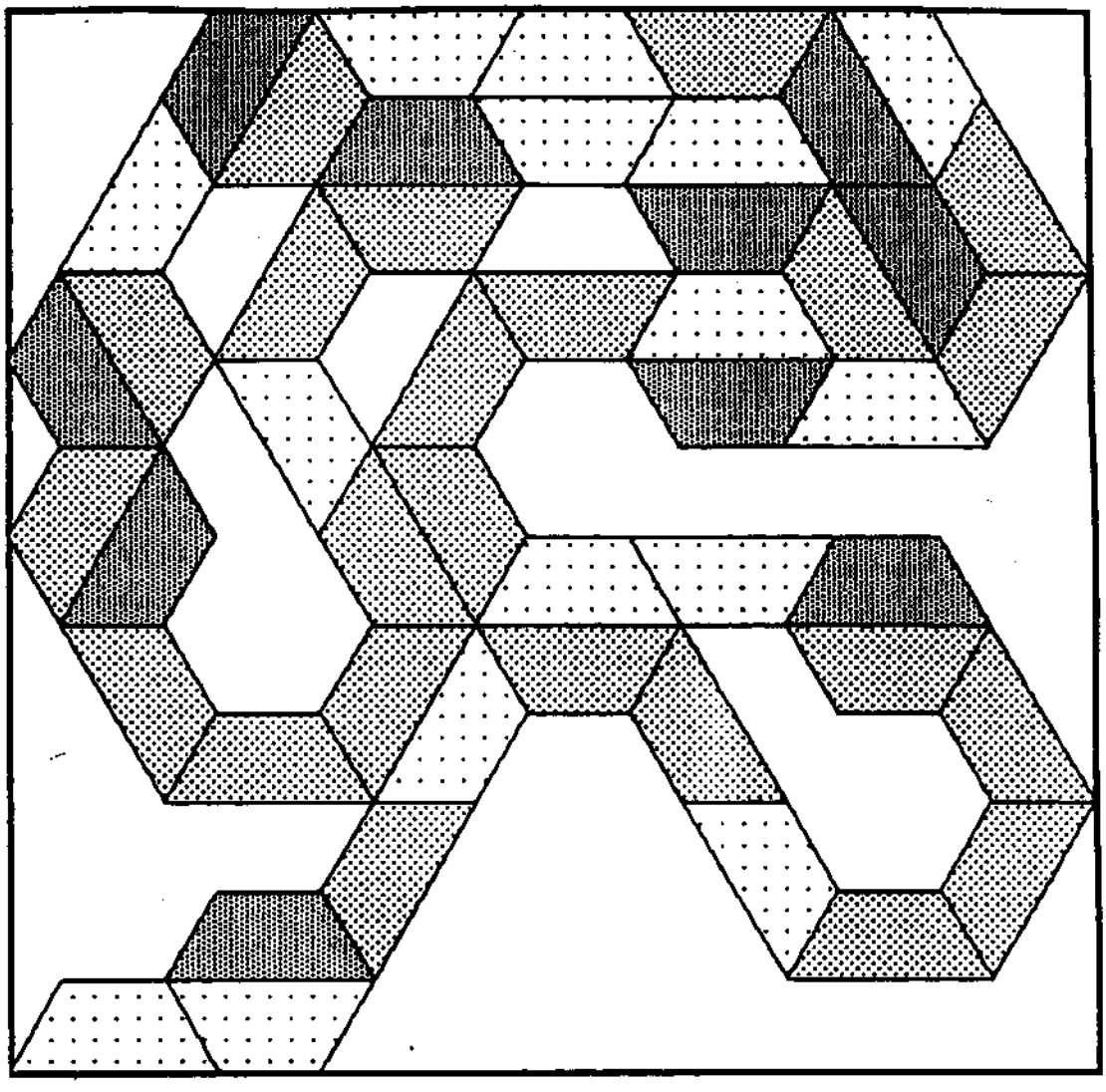


**Figure 3** 5X5 box with 48 half-hexagons

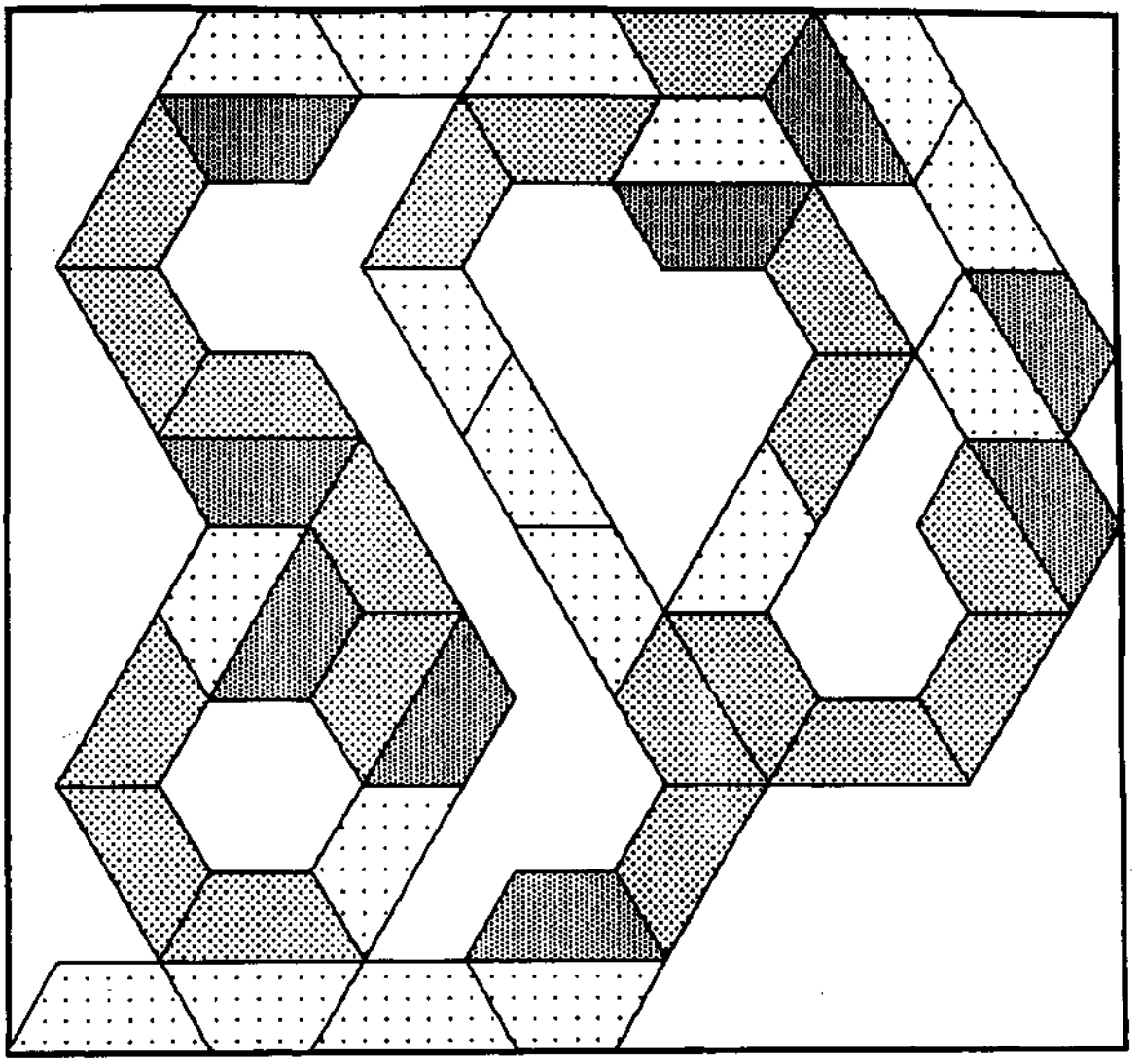
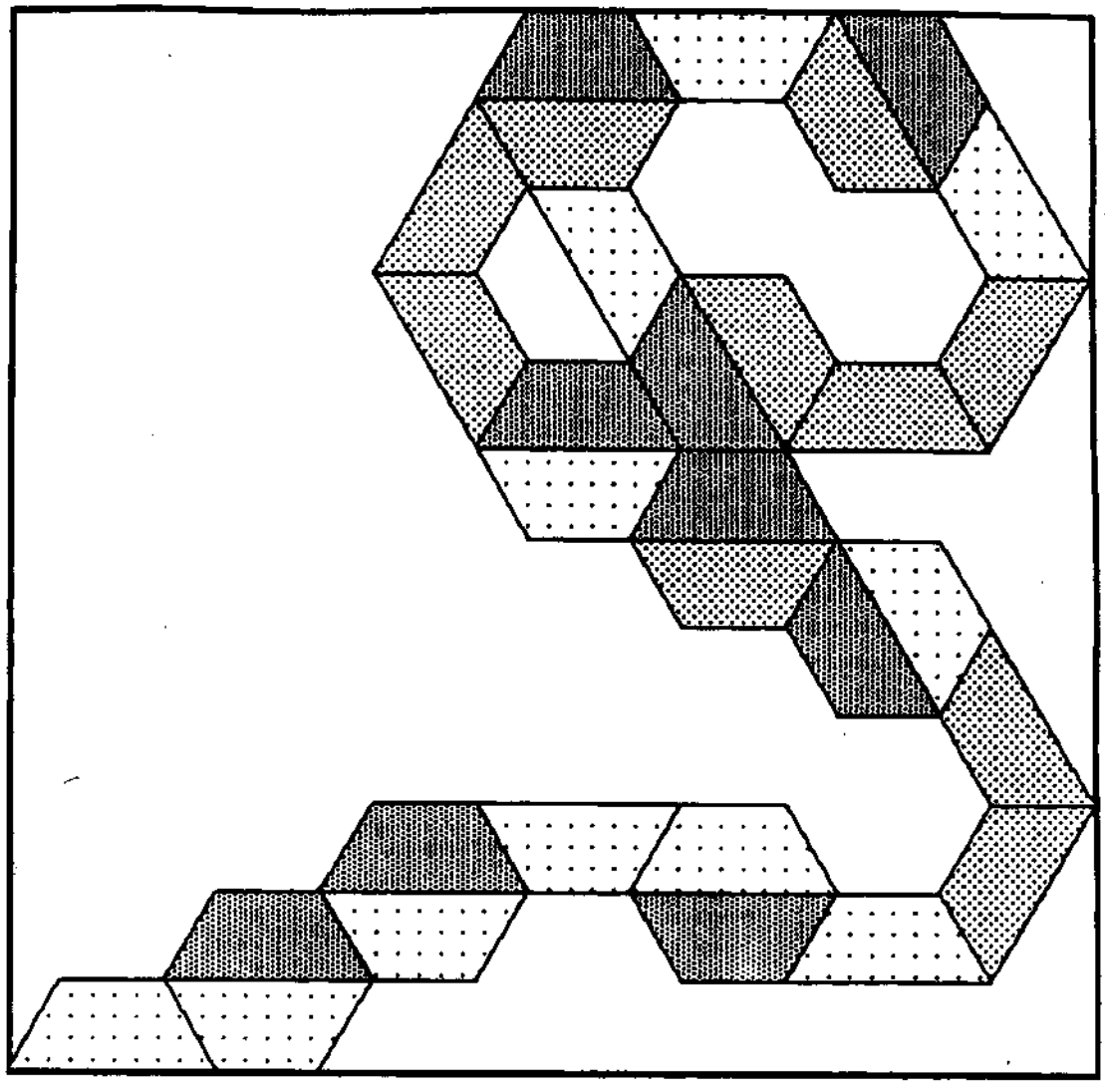


**Figure 4** 5X5 box with 45 half-hexagons



m e 5a Tran t sol with 30 half-hexagons evolving to Figure 5b

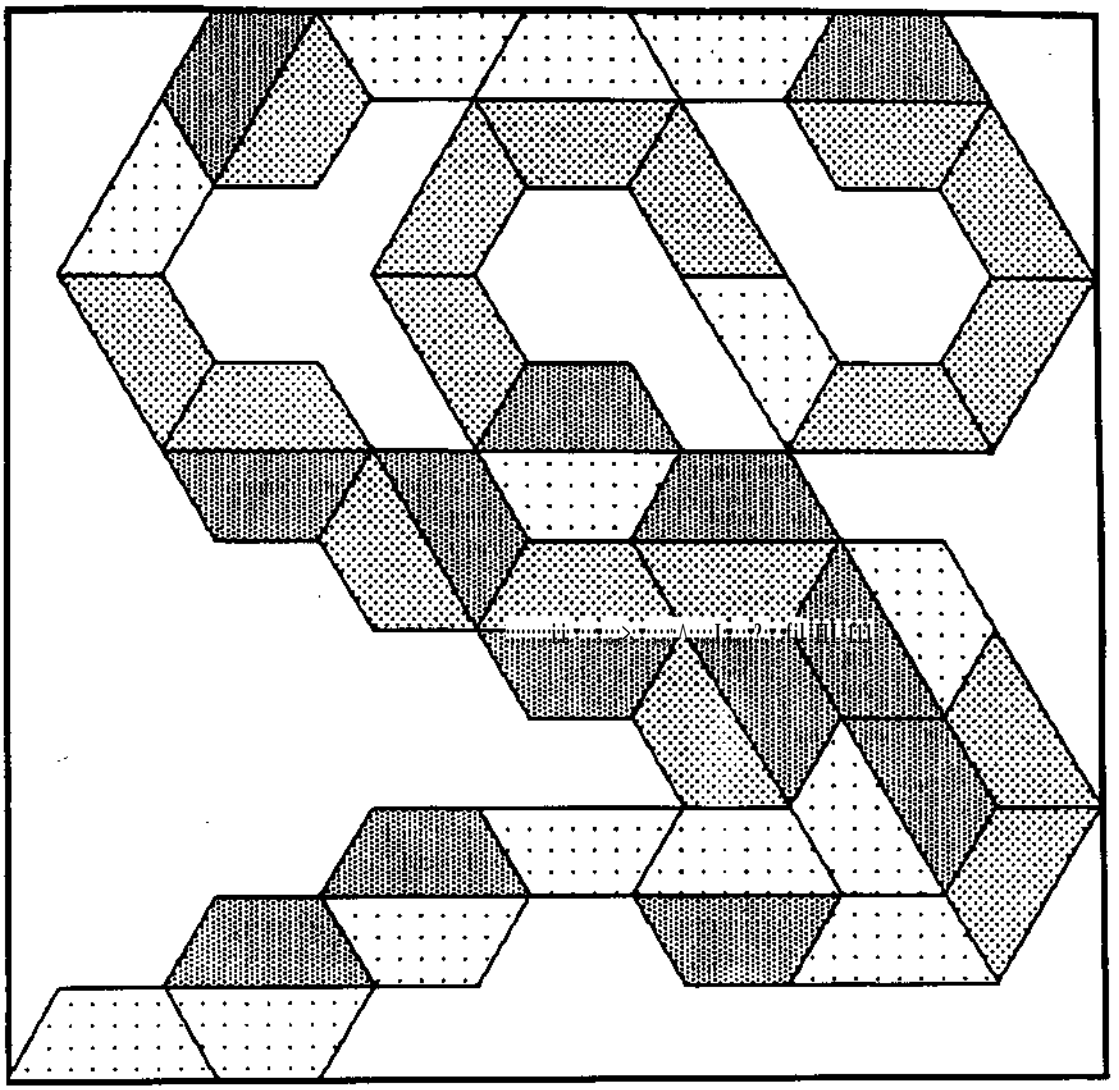


**Figure 5b** 5X5 box with 44 half-hexagons

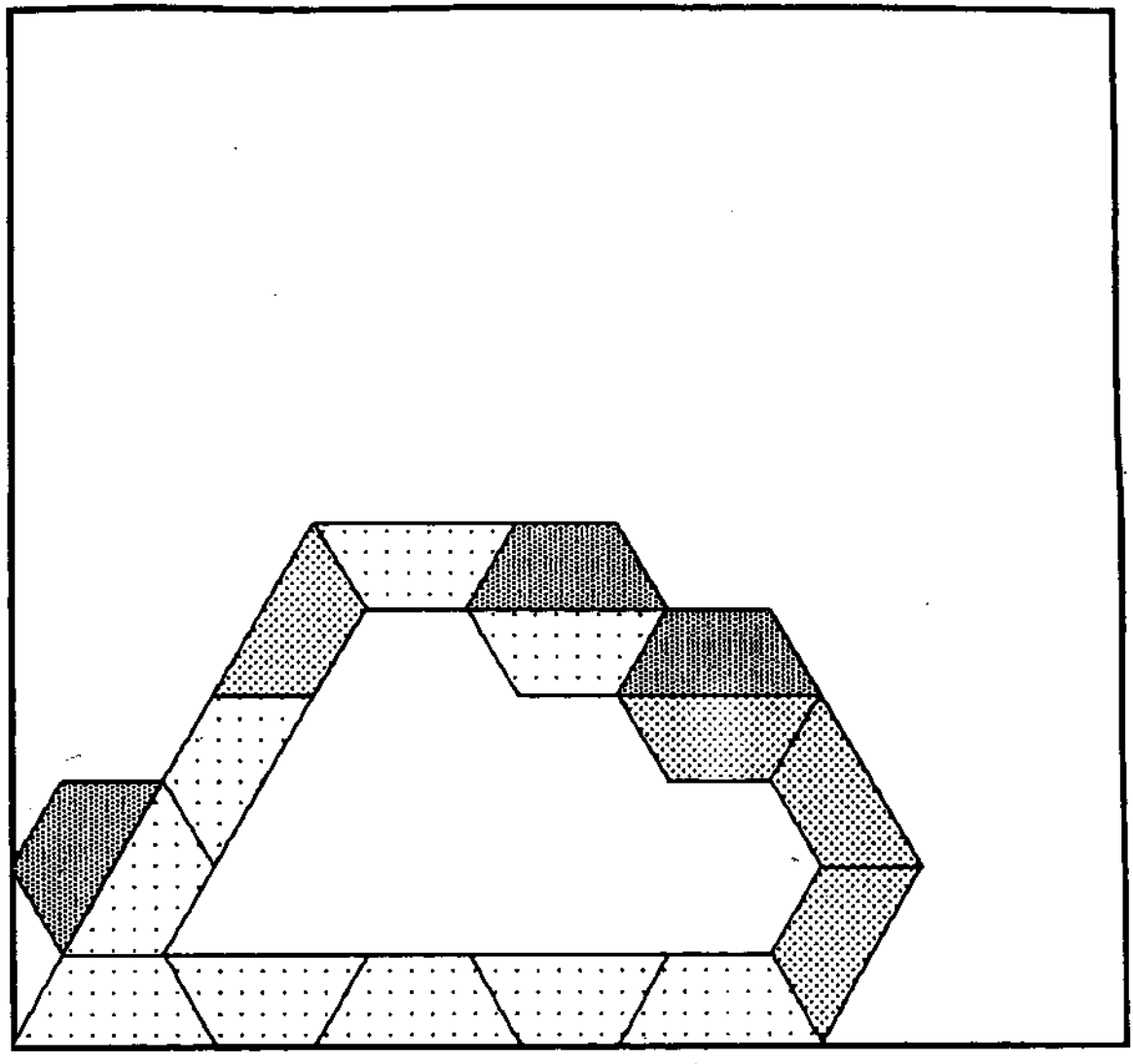


**Figure 6a** Transient solution with 16 half-hexagons evolving to Figure 5b

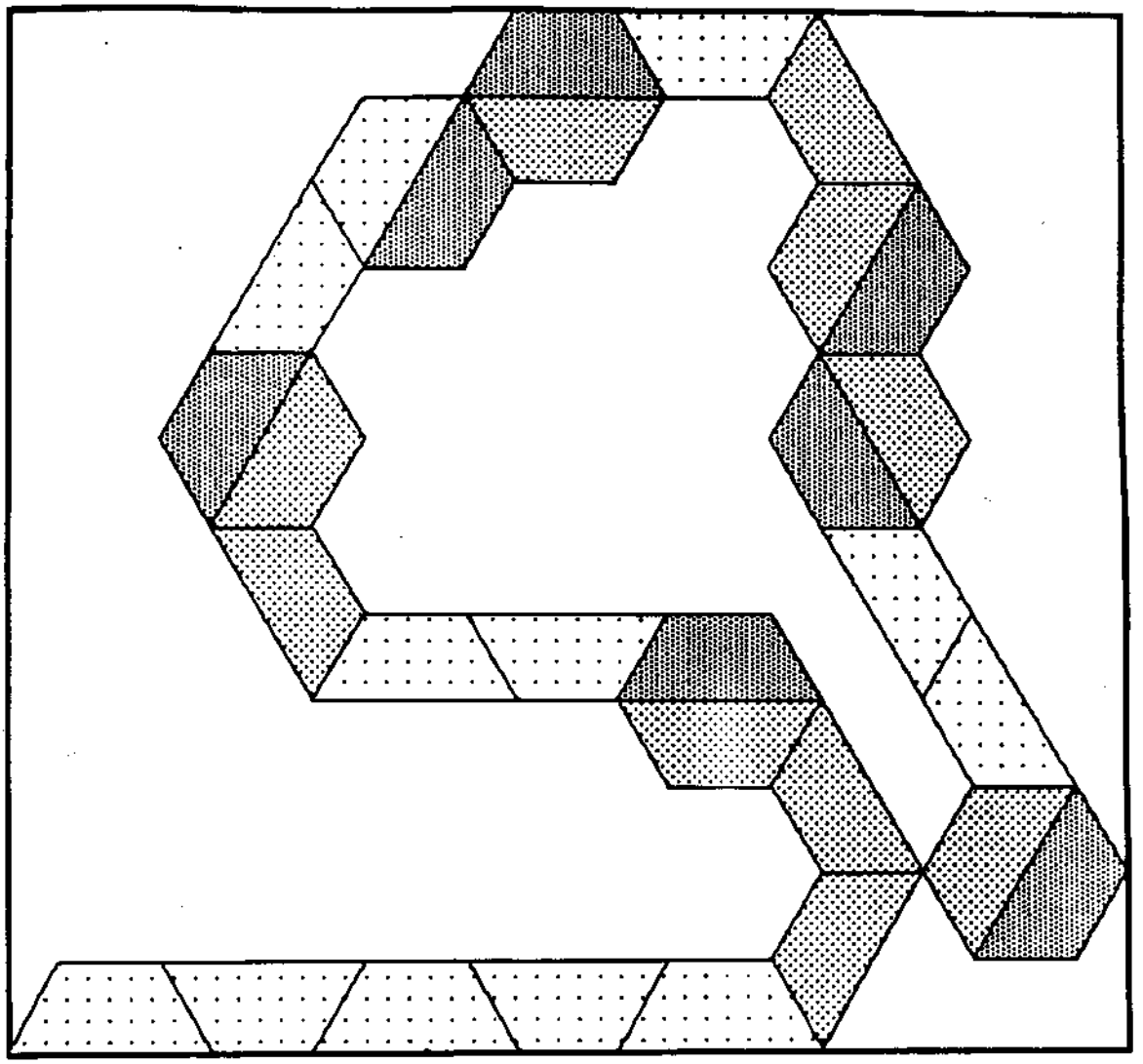


**Figure 6b 5X5 box with 29 half-hexagons**

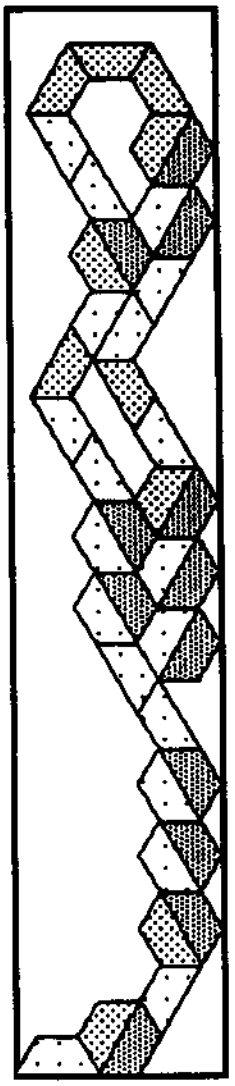


**Figure 7** 2X12.5 rectangle with 39 half-hexagons

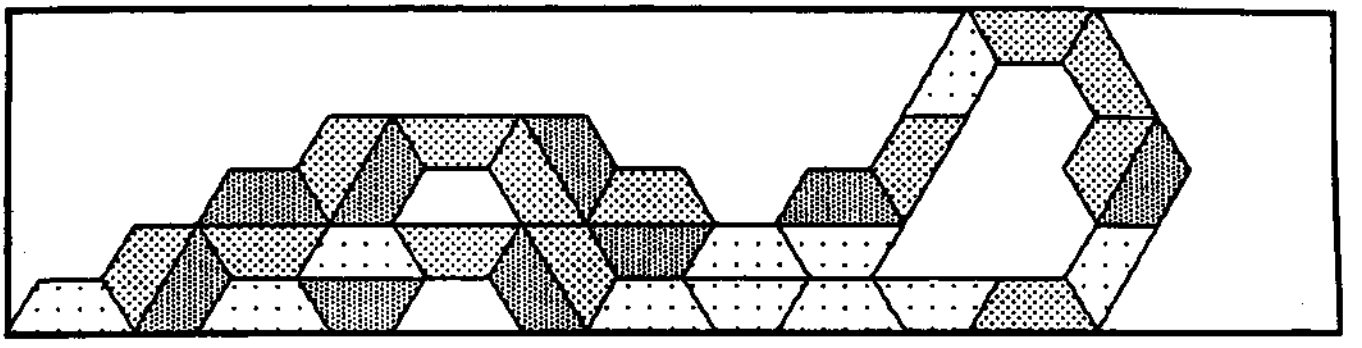


**Figure 8 10X2.5 rectangle with 33 half-hexagons**

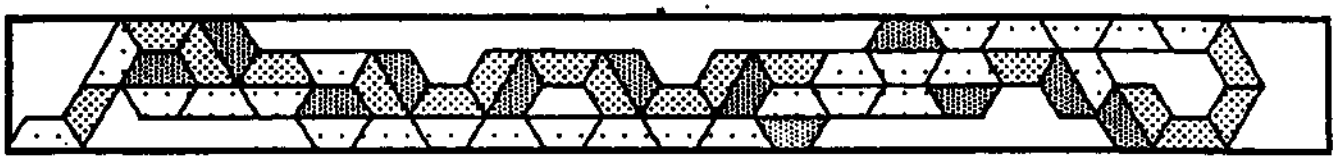


**Figure 9 16.67X1.5 rectangle with 55 half-hexagons**

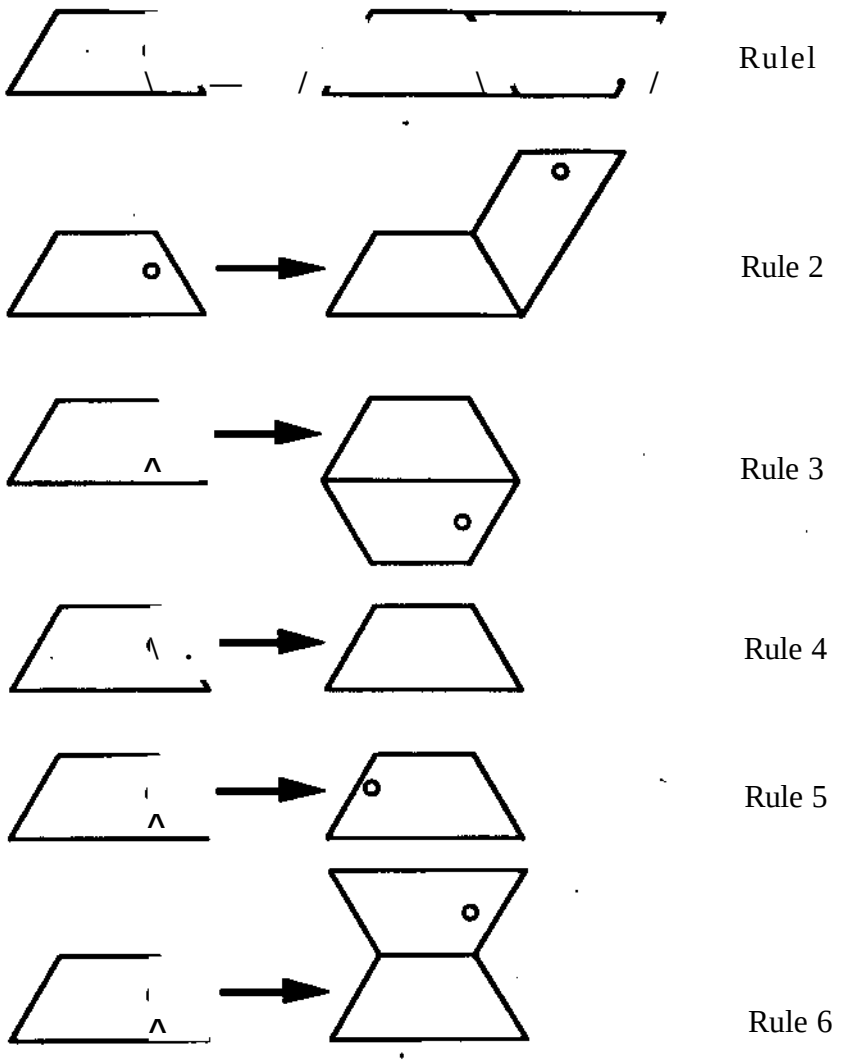



**Figure 10** Extended Shape Grammar