AD/A-001 814

SHAPE GRAMMARS AND THEIR USES

James Gips

Stanford University

Prepared for:

Advanced Research Projects Agency

March 1974

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD/A021814

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: STAN-CS-74-413	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SHAPE GRAMMARS AND THEIR USES.	5. TYPE OF REPORT & PERIOD COVERED: technical, March 1974
6. PERFORMING ORG. REPORT NUMBER: STAN-CS-74-413	
7. AUTHOR(s): James Gips	8. CONTRACT OR GRANT NUMBER(s): DAHC 15-73-C-0435
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Stanford University, Computer Science Department, Stanford, California 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: ARPA/IPT, Attn: S. D. Crocker, 1400 Wilson Blvd., Arlington, Va. 22209	12. REPORT DATE: March, 1974
	13. NUMBER OF PAGES: 243
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): ONR Representative: Philip Surra, Durand Aeronautics Bldg., Rm. 165, Stanford University, Stanford, California 94305	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Releasable without limitations on dissemination.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Shape grammars are defined and their uses are investigated. Shape grammars provide a means for the recursive specification of shapes. A shape grammar is presented that generates a new class of reversible figures. Shape grammars are given for some well known mathematical curves. A simple method for constructing shape grammars that simulate Turing machines is presented. A program has been developed that uses a shape grammar to solve a perceptual task involving the analysis and comparison of line drawings that portray three-dimensional objects of a restricted type. A formalism

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

(continued

1. Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

that uses shape grammars to generate paintings is defined, its implementation on the computer is described, and examples of generated paintings are shown. The use of shape grammars in generating paintings has led to an investigation of aesthetics. A formalism for specifying aesthetic viewpoints is described. A program has been written that evaluates the paintings generated using shape grammars in terms of a well-defined aesthetic viewpoint specified with this formalism.

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO AIM-231

MARCH 1974

COMPUTER SCIENCE DEPARTMENT
REPORT NO. STAN-CS-74-413

SHAPE GRAMMARS AND THEIR USES

by

JAMES GIPS

ABSTRACT: Shape grammars are defined and their uses are investigated. Shape grammars provide a means for the recursive specification of shapes. A shape grammar is presented that generates a new class of reversible figures. Shape grammars are given for some well known mathematical curves. A simple method for constructing shape grammars that simulate Turing machines is presented. A program has been developed that uses a shape grammar to solve a perceptual task involving the analysis and comparison of line drawings that portray three-dimensional objects of a restricted type. A formalism that uses shape grammars to generate paintings is defined, its implementation on the computer is described, and examples of generated paintings are shown. The use of shape grammars in generating paintings has led to an investigation of aesthetics. A formalism for specifying aesthetic viewpoints is described. A program has been written that evaluates the paintings generated using shape grammars in terms of a well-defined aesthetic viewpoint specified with this formalism.

This research was supported in part by a fellowship from the International Business Machines Corporation and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract No. SD-183 and Contract No. DAHC 15-73-C-0435.

ACKNOWLEDGMENTS

Important parts of this research were done in collaboration with George Stiny, a graduate student in Systems Science at U.C.L.A. In particular, the three formalisms used (i.e. shape grammars, the formalism for specifying paintings, and the formalism for specifying aesthetic viewpoints) were developed jointly. Sections 1.1 and 3.1.1.1 - 3.1.1.4 are a revised version of [Stiny and Gips 1972]. Portions of Section 3.2 are expansions of [Gips and Stiny 1973], [Stiny and Gips 1974], and [Gips and Stiny, submitted for publication]. Stiny's research has concentrated on the mathematical properties of these formalisms, rather than their applications, and will be reported in his forthcoming dissertation [Stiny, in preparation].

I would like to thank Professor Jerome Feldman for his invaluable help as my thesis advisor and his consistently good advice throughout my stay at Stanford, Professors Jeffrey Raskin and Vinton Cerf for serving so conscientiously on my reading committee, and Professors Thomas Cover, John McCarthy, and Robert Floyd (as well as those previously named) for valuable discussions on various facets of this research.

Finally, I would like to thank my wife, Pat, for her constant support, understanding, and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

(* indicates figure contains a shape grammar)

Section 1

Section 2

Section 3

NOTE: Pages iii and 226 were not printed.
Your copy is complete without these pages.

INTRODUCTION

Shape grammars provide a means for the recursive specification of shapes. The formalism for shape grammars is designed to be easily usable and understandable by people and at the same time to be adaptable for use in computer programs.

Shape grammars are similar to phrase structure grammars, which were developed by Chomsky [1956, 1957]. Where a phrase structure grammar is defined over an alphabet of symbols and generates a language of sequences of symbols, a shape grammar is defined over an alphabet of shapes and generates a language of shapes.

This dissertation explores the uses of shape grammars. The dissertation is divided into three sections and an appendix.

In the first section: Shape grammars are defined. Some simple examples are given for instructive purposes. Shape grammars are used to generate a new class of reversible figures. Shape grammars are given for some well-known mathematical curves (the Snowflake curve, a variation of Peano's curve, and Hilbert's curve). To show the general computational power of shape grammars, a procedure that given any Turing machine constructs a shape grammar that simulates the operation of that Turing machine is presented. Related work on various formalisms for picture grammars is described. A symbolic characterization of shape grammars is given that is useful for implementing shape grammars in computer programs.

In the second section, a program that uses a shape grammar to solve a perceptual task is described. The task involves analyzing and comparing line drawings that portray three-dimensional objects of a restricted type.

The third section is divided into two parts. First, a formalism for generating paintings is defined. The primary component of this formalism is a shape grammar. The paintings generated are material representations of shapes specified by shape grammars. The computer implementation of this formalism is described. The second part is concerned with aesthetics. A formalism is defined for specifying an aesthetic viewpoint. The formalism is used to specify a particular aesthetic viewpoint for interpreting and evaluating paintings generated using shape grammars. This viewpoint has been implemented on the computer. The net result is that the program described in Section 3 can be used to interactively define the rules for producing a painting, can use the rules to generate and display the resulting painting, and can then evaluate the painting relative to the specific aesthetic viewpoint. Relationships between the formalism for aesthetic viewpoints and information theory, science, and Meta-Dendral [Buchanan et. al. 1971] are touched upon. Finally, the possibility of using this approach to aesthetics to write programs that automatically analyze presented art objects or design new art objects is explored.

In the Appendix, a method for constructing the inverse of a Turing machine is presented. This construction was created in response to a problem that is described in the aesthetics section.

This dissertation ranges over many subjects -- perceptual figures,

mathematical curves, Turing machines, painting, aesthetics. Presumably, this reflects the wide variety of applications of shape grammars.

SECTION 1 SHAPE GRAMMARS: DEFINITIONS, EXAMPLES, RELATED WORK

1.1 Definitions

A shape grammar, SG, is a 4-tuple : SG = $\langle V_t, V_m, R, I \rangle$ where

(1) V_t is a finite set of shapes.

(2) V_m is a finite set of shapes such that $V_t \cap V_m = \phi$.

(3) R is a finite set of ordered pairs (u,v) such that u is a shape consisting of an element of V_t^* combined with an element of V_m^* and v is a shape consisting of an element of V_t^* combined with an element of V_m^*.

(4) I is a shape consisting of an element of V_t^* combined with an element of V_m^*.

Elements of the set V_t are called terminal shape elements (or terminals). Elements of the set V_m are called non-terminal shape elements (or markers). The sets V_t and V_m must be disjoint. Elements of the set V_t^* are formed by the finite arrangement of one or more elements of V_t in which any elements and/or their mirror images may be used a multiple number of times in any location, orientation, or scale. The set $V_t^* = V_t^* \cup \{\epsilon\}$ where ϵ is the empty shape. The sets V_m^* and V_m^* are defined similarly. Elements (u,v) of R are called shape rules and are written $u \rightarrow v$. u is called the left side of the rule; v the right side of the rule. u and v usually are enclosed in identical dotted rectangles to show the correspondence between the two shapes. I is called the initial shape and normally contains a u such that there is a (u,v) which is an element of R.

A shape is generated from a shape grammar by beginning with the initial shape and recursively applying the shape rules. The result of applying a shape rule to a

given shape is another shape consisting of the given shape with the right side of the rule substituted in the shape for an occurrence of the left side of the rule. Rule application to a shape proceeds as follows :

> (1) Find part of the shape that is geometrically similar to the left side of a rule in terms of both terminal and non-terminal elements. There must be a one-to-one correspondence between the terminals and non-terminals in the left side of the rule and the terminals and non-terminals in the part of the shape to which the rule is to be applied.
>
> (2) Find the geometric transformations (scale, translation, rotation, mirror image) which make the left side of the rule identical to the corresponding part in the shape.
>
> (3) Apply those transformations to the right side of the rule.
>
> (4) Substitute the transformed right side of the rule for the part of the shape which corresponds to the left side of the rule.

The generation process is terminated when no rule in the grammar can be applied.

For any given shape grammar, the dimensionality of the shapes in V_m and V_t and of the geometric transformations used to combine these shapes must be constant. This number is called the dimensionality of the shape grammar. While three-dimensional shape grammars have been used to generate sculpture [Stiny and Gips 1972], in this report only two-dimensional shape grammars are considered. All elements of V_t and V_m will be two-dimensional and all transformations will be planar.

The sentential set of a shape grammar, SS(SG), is the set of shapes (sentential shapes) which contains the initial shape and all shapes which can be generated from the initial shape using the shape rules. The language of a shape grammar, L(SG), is the set of sentential shapes that contain only terminals, i.e., $L(SG) = SS(SG) \cap V_t^*$.

In this definition of shape grammars, shapes and the transformations on shapes are used as primitives. This enables shape grammars to be visually oriented and facilitates their use. A more traditional symbolic characterization of shape grammars that uses symbols and functions as primitives is given in section 1.9.

1.2 Restricted shape grammars

Just as with phrase structure grammars [Hopcroft and Ullman 1969], types of shape grammars can be defined by putting further restrictions on the allowable form of shape rules. Two types of shape grammars, non-erasing shape grammars and unimarker shape grammars, are especially useful.

A non-erasing shape grammar is a shape grammar in which all terminal elements that appear in the left side of each rule appear identically in the right side of that rule. The result of this restriction is that once a terminal is added during the generation process using a non-erasing shape grammar, it canot be erased.

A unimarker shape grammar is a non-erasing shape grammar in which the initial shape contains exactly one marker, the left side of each rule contains exactly one marker, and the right side of each rule contains zero or one markers. The result of this restriction is that each sentential shape of a unimarker shape grammar that is not in the language of the shape grammar contains exactly one marker.

This topic is explored in detail in [Stiny, in preparation].

1.3 Parallel shape grammars

Shape generation as described in Section 1.1 is based on the serial application of shape rules, i.e., at each step of the generation a shape rule is applied to only one part of a shape. Parallel generation using shape grammars is also possible. In the parallel generation of a shape, whenever a shape rule is used, it is applied simultaneously to every part of the shape to which it is applicable. A shape grammar which is intended to be used in this manner is called a parallel shape grammar. There is no formal difference between a parallel shape grammar and a (serial) shape grammar; it is simply a matter of the intended method of rule application.

Parallel string grammars were first defined and investigated by Rosenfeld [1971]. As with string grammars, the same shape grammar can be used both in serial and in parallel, but the two languages defined may differ. Examples of this phenomenon are given in Sections 1.4 and 1.6.1.

Hereafter, a shape grammar that is intended to be used in parallel will be denoted PSGn where n is the number of the shape grammar; a shape grammar intended to be used in serial will be denoted SGn. Shape grammars will be assumed to be used in serial unless noted otherwise.

1.4 Examples: embedded squares

In this section, four simple, related shape grammars are examined for pedagogical purposes.

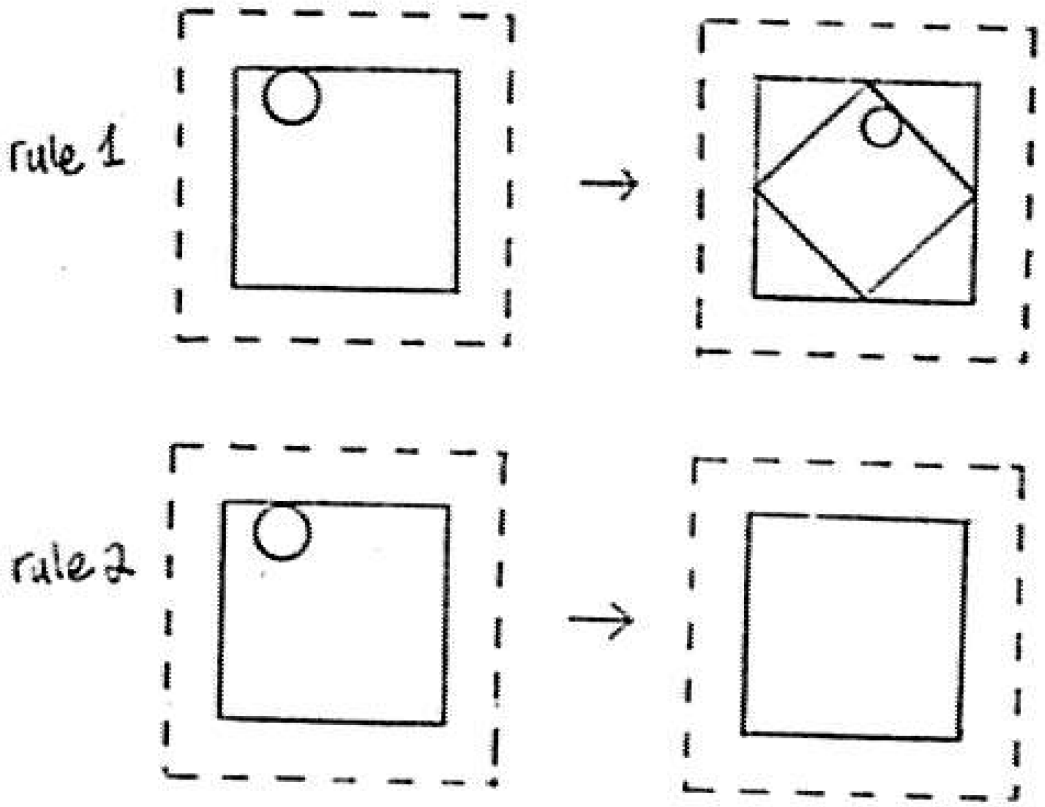
A very simple (unimarker) shape grammar, SG1, is shown in Figure 1a. V_t contains a square as its only element. V_m contains a circle as its only element. All sentential shapes will be composed of squares and/or circles. All shapes in the language will be composed of squares only. There are two shape rules. The dotted rectangles around the left and right side of each shape rule indicate the correspondence between the two shapes. The initial shape contains a square and an attached circle.

The generation of a shape in L(SG1) is shown in Figure 1b. Because the two shape rules in SG1 contain identical left sides, the two shape rules are applicable in identical circumstances, i.e., wherever there is a square with attached circle geometrically similar to the shapes on the left side of each rule. Application of rule 1 to a shape results in the addition of an embedded square and the shrinking and moving of the marker. Application of rule 1 forces the continuation of the generation process as both rules are applicable to the new sentential shape. Application of rule 2 to a shape results in the removal of the marker, thereby halting the generation process as no rules are now applicable, and yields a shape in the language. In the generation shown in Figure 1b, the process is begun with the initial shape, rule 1 is applied three times, and then rule 2 is applied. The language defined by SG1 is shown in Figure 1c.

$SGI = \langle V_T, V_M, R, I \rangle$

$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains



I is

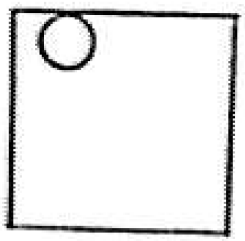


Figure 1a. SGI, a shape grammar for inscribed squares.

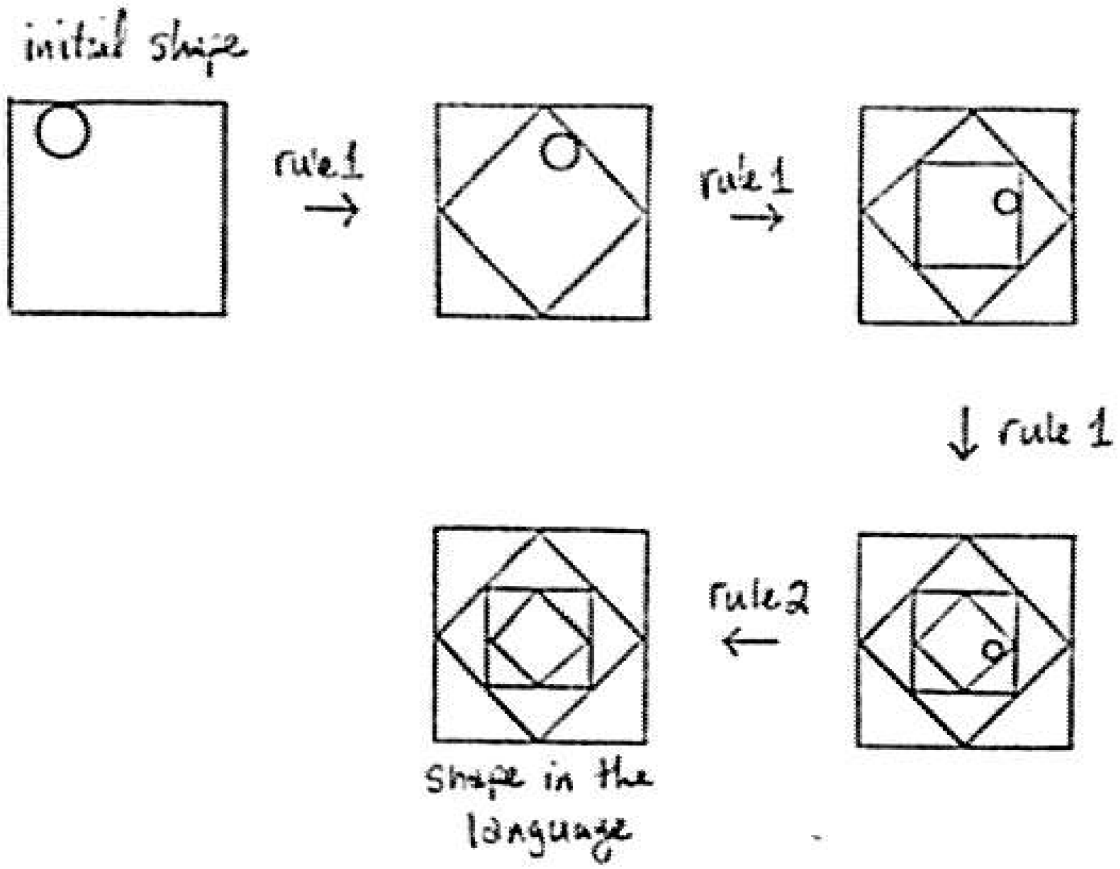


Figure 1b. A generation using SGI.

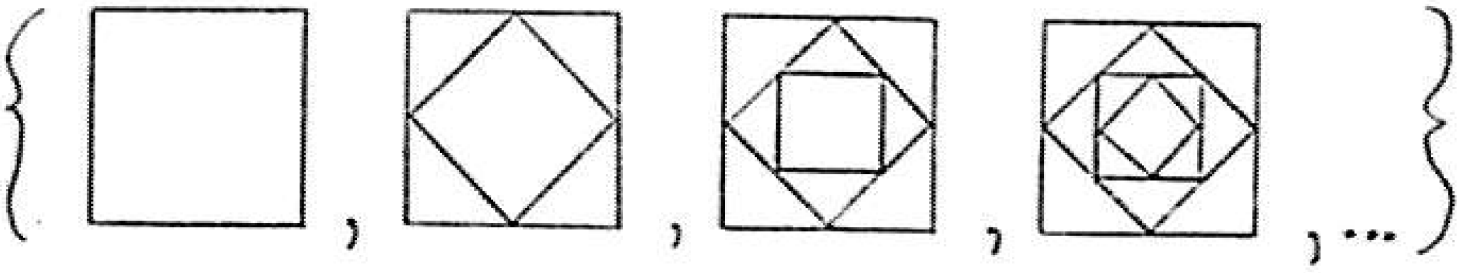


Figure 1c. The language defined by SGI.

A somewhat similar shape grammar, SG2, is shown in Figure 2a. The generation of a shape using SG2 is shown in Figure 2b. Where in the generation process using SG1 squares are succesively inscribed, using SG2 squares are successively circumscribed. The language defined by SG2 is shown in Figure 2c. Note that the area contained in the shapes in L(SG1) is constant, where the area contained in successive shapes in L(SG2) doubles.

The purpose of the marker in these two examples may not be apparent. The use of the marker makes the rules applicable only to the most recently added square. If the marker were not used, rule 1 could be applied over and over to the same square. The importance of markers is further illustrated by the next two examples.

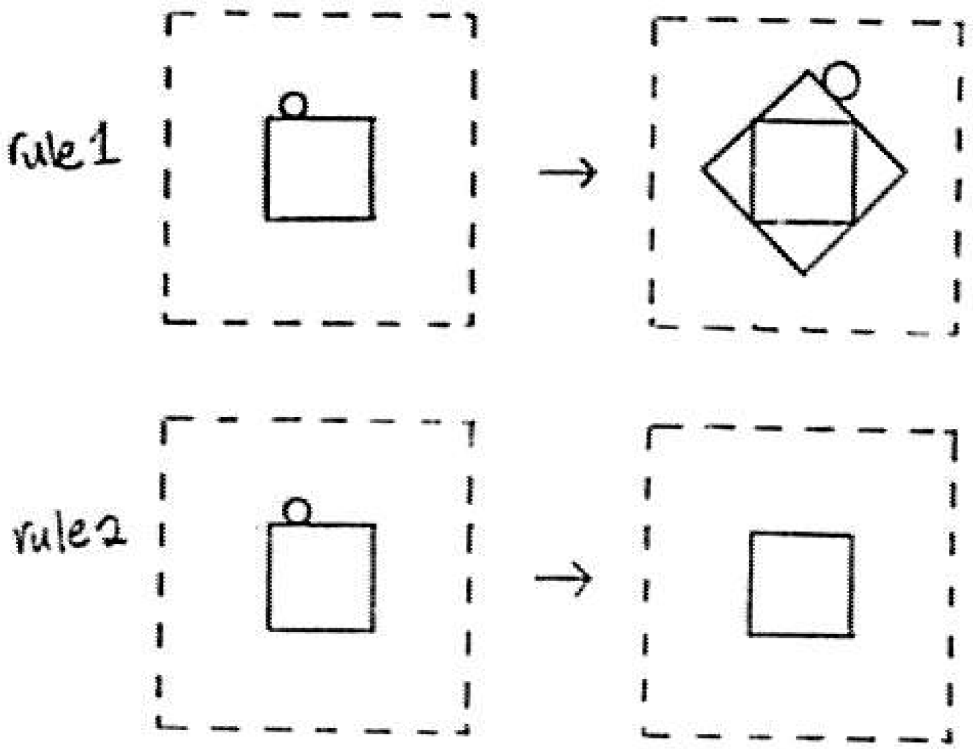
The shape grammar SG3, shown in Figure 3a, is similar to SG1 but embeds four squares instead of one. A generation using SG3 is shown in Figure 3b. Each application of rule 1 causes four terminals (squares) to be added. Because the right side of the first shape rule of SG3 contains only one marker, only one square at each level can be expanded. The relative location of the marker in the right side of rule 1 determines the exact sequence of subsquares to which the marker is attached in the generation and therefore which square at each level can be expanded. The language generated by SG3 is shown in Figure 3c.

In the shape grammar SG4, shown in Figure 4a, the right side of rule 1 contains four markers. This rule allows squares to be embedded subsequently in any of the four added squares. A generation of a shape using SG4 is shown in Figure 4b. Each application of rule 1 causes four markers (circles) and terminals (squares) to

$SG2 = \langle V_T, V_M, R, I \rangle$

$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains



I is

Figure 2a. SG2, a shape grammar for circumscribed squares.

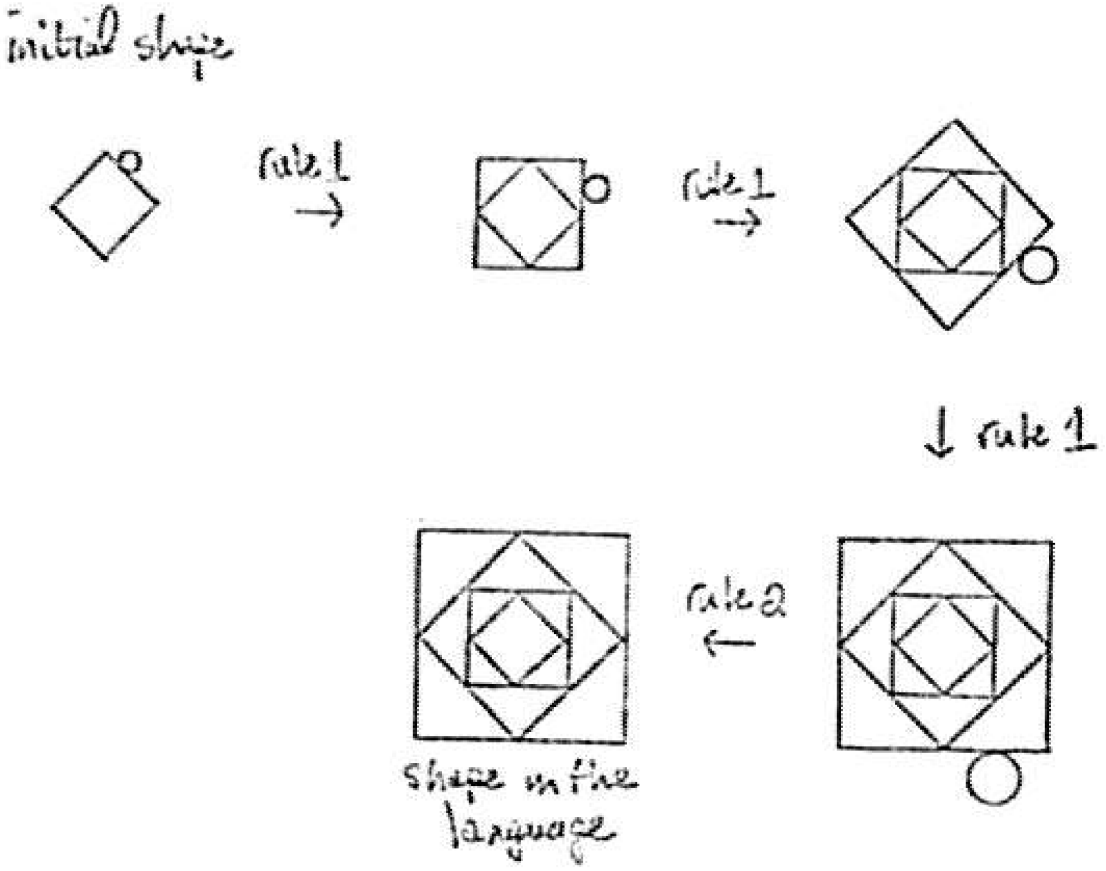


Figure 2b. A generation using SG2.

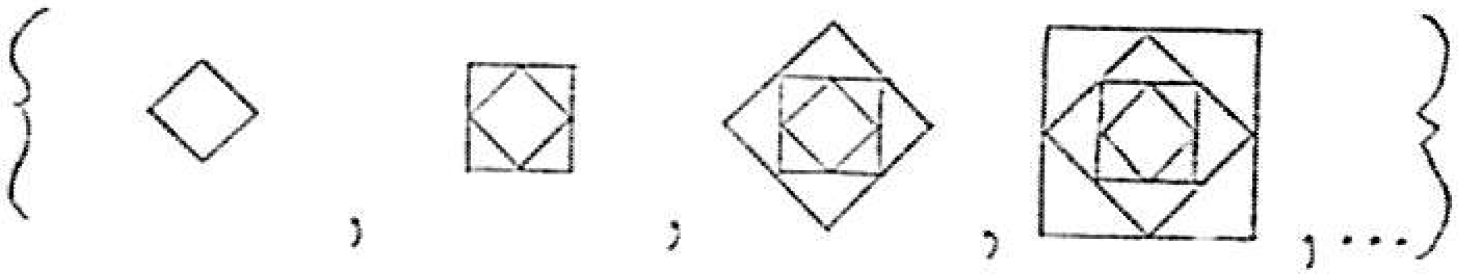
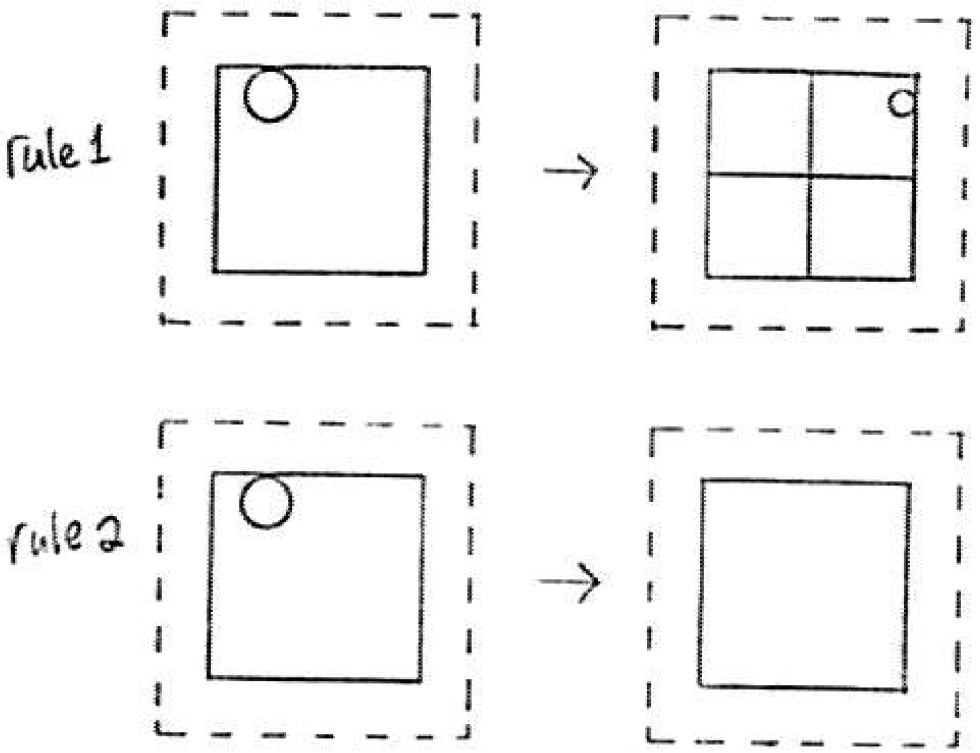


Figure 2c. The language defined by SG2.

$SG3 = \langle V_T, V_M, R, I \rangle$

$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains



I is

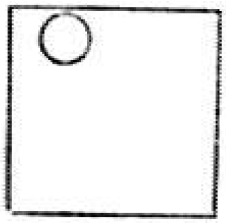


Figure 3a. SG3, a shape grammar for labelled squares.

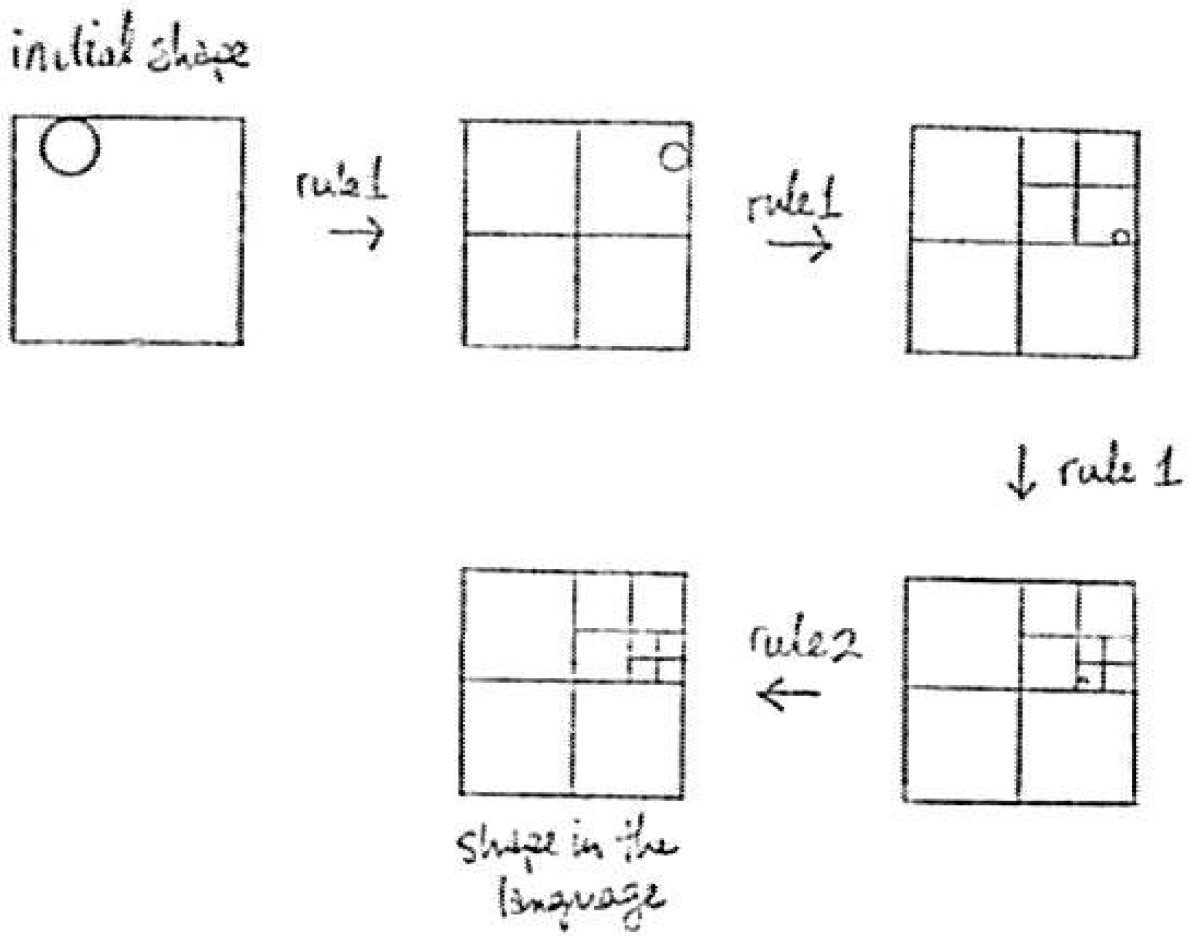


Figure 3b. A generation using SG3.

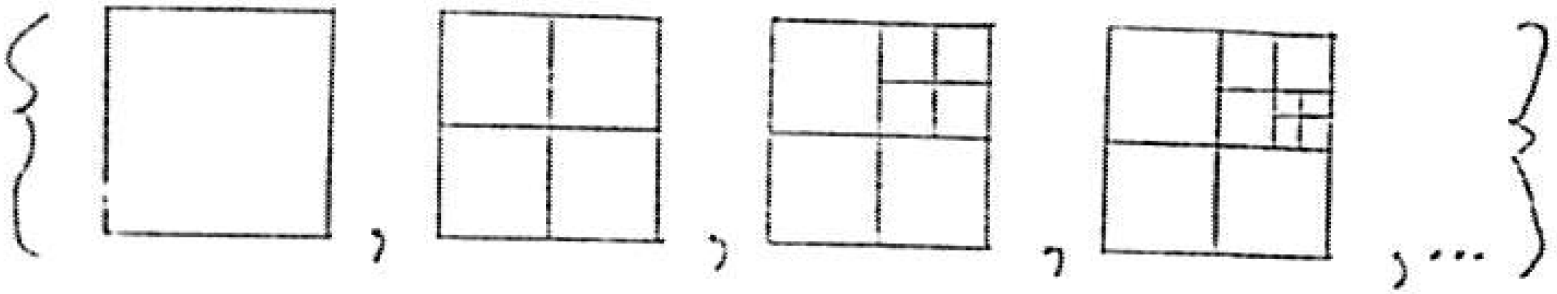


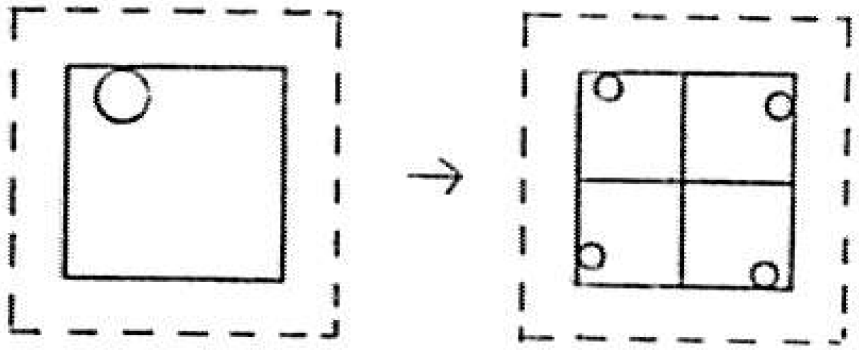
Figure 3c. The language defined by SG3.

$SG4 = \langle V_T, V_M, R, I \rangle$

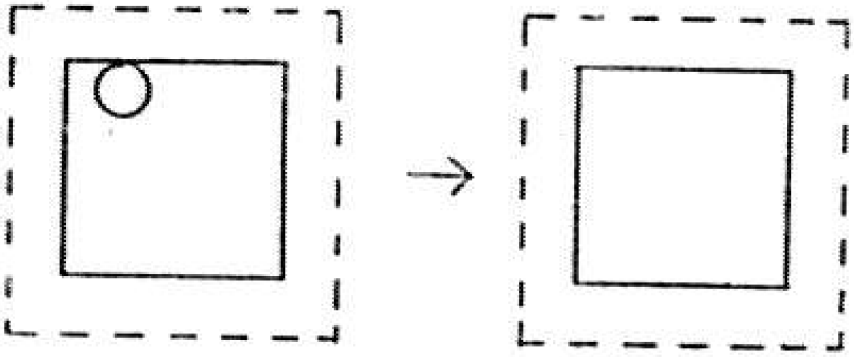
$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R Contains

Rule 1



Rule 2



I is

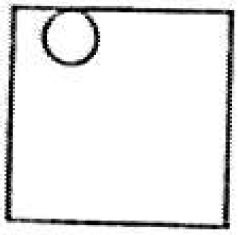


Figure 4a. SG4, another shape grammar for embedded squares.

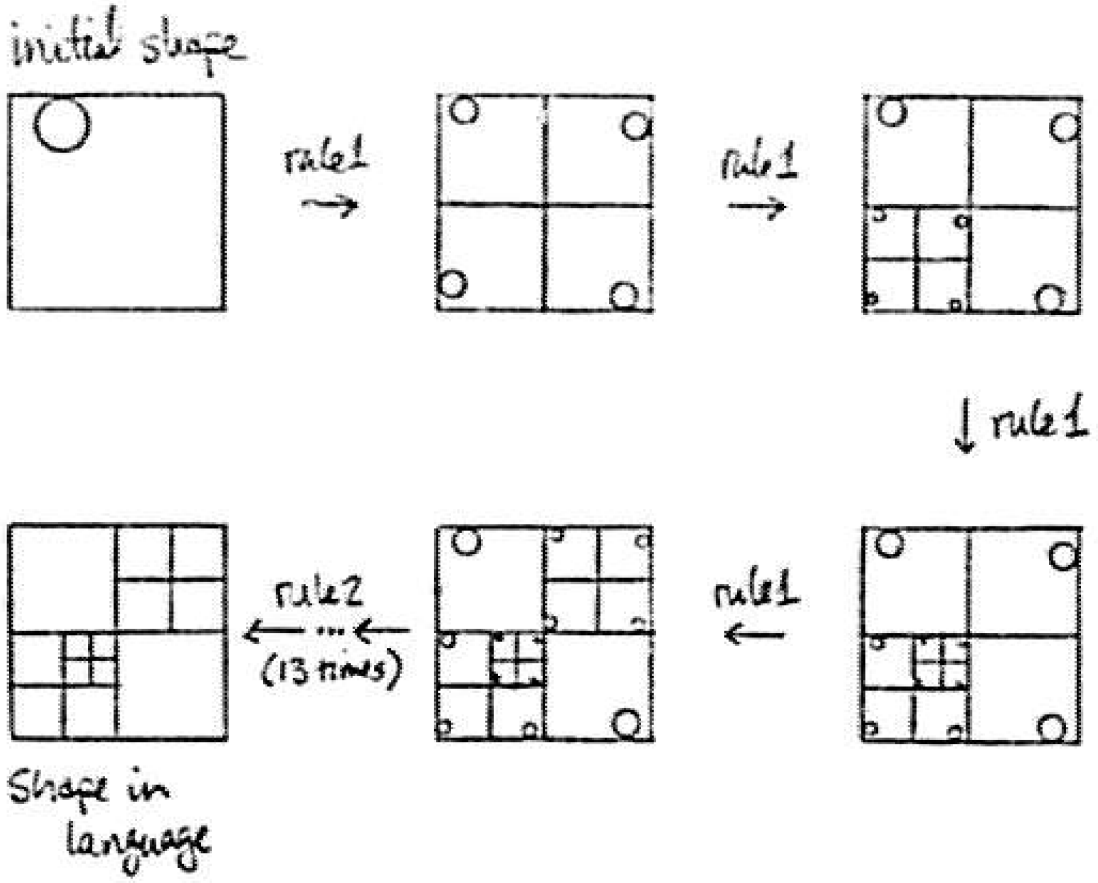


Figure 4b. A generation using SG4 in serial.

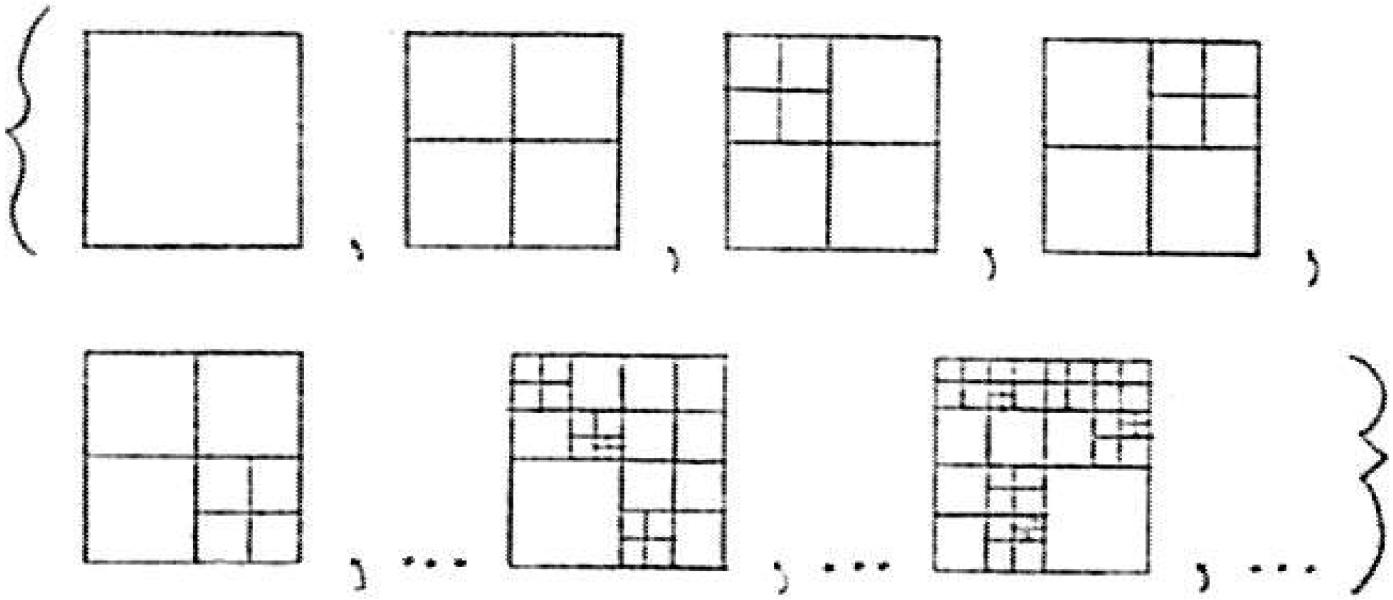


Figure 4c. The language defined by SG4 used in serial.

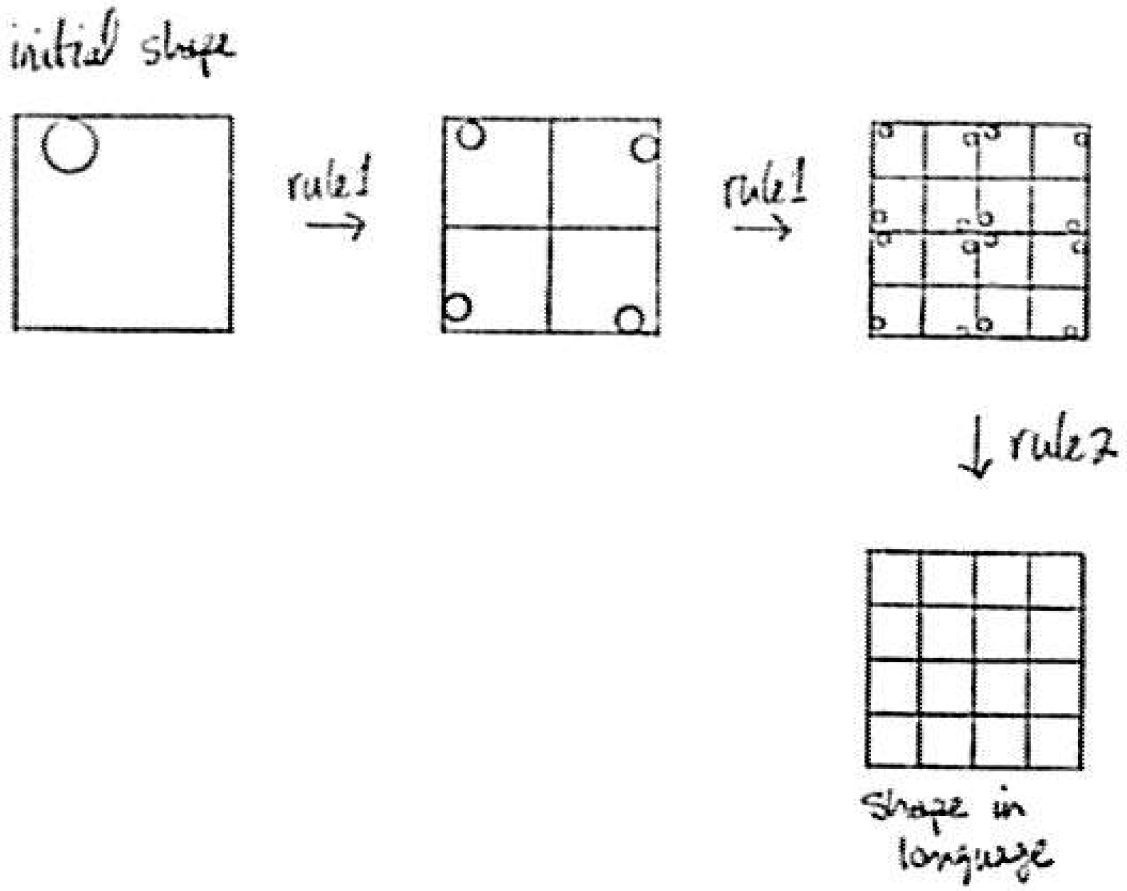


Figure 4d. A generation using SG in parallel.

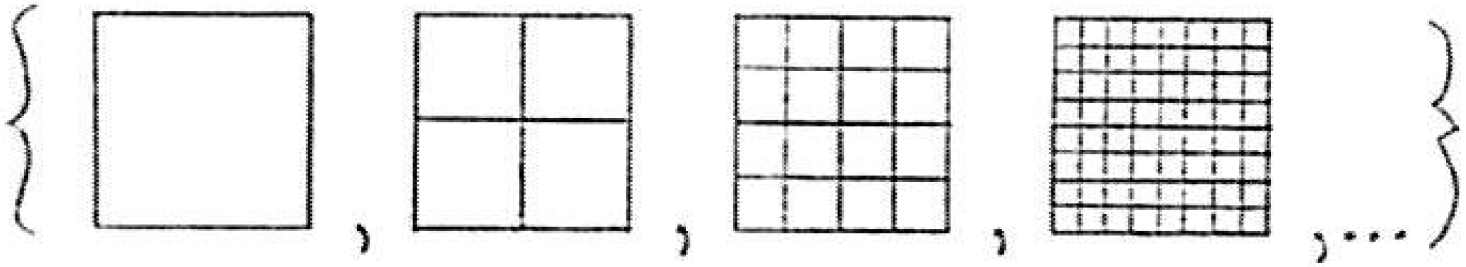


Figure 4e. The language defined by SG used in parallel.

be added to the shape and one marker to be erased. Generation using SG4 can proceed to an arbitrary depth in any part of the shape. L(SG3) is a small subset of the language generated by SG4 (see Figure 4c). Similar languages of embedded squares can be generated using shape grammars with different configurations of markers. Markers are important because they restrict rule application to specific parts of a shape and help determine the transformations (e.g. scale) required to apply the rules.

The shape grammar SG4 can also be used in parallel. Recall that in a parallel generation whenever a shape rule is used it is applied simultaneously to every part of the shape to which it is applicable. In a parallel generation using SG4, whenever rule 1 is used every circle attached to a square is expanded simultaneously. Similarly, whenever rule 2 is used every circle attached to a square is erased simultaneously. The result is that in the generation of a shape using SG4 in parallel, the first application of rule 1 causes four markers and terminals to be added, the second application causes sixteen to be added, the third sixty four, etc. A generation using SG4 in paralel is shown in Figure 4d. The language defined by SG4 used in parallel is shown in Figure 4e. The language defined by SG4 is essentially a sequence of successively fine square grids. Each shape in the language defined using SG4 in parallel has been expanded uniformly throughout. Note that this language is a small subset of the language defined by SG4 used in serial.

1.5 Example: reversible figure

A new reversible figure [Gips 1972], similar to the Necker cube, the Schroeder reversible staircase, etc. [Luckiesh 1965] is shown in Figure 5a. The figure can represent two different three-dimensional objects. The central three lines can be perceived either as outer (convex) edges of a cube or as inner (concave) edges where a cube was cut from the closest corner of a larger cube. Either the outer walls of the object appear to have width and be solid or the outer walls appear to have no width and be infinitely thin. A variation is shown in Figure 5b.

A parallel shape grammar, PSG5, that generates these figures is shown in Figure 6a. A generation using this shape grammar is shown in Figure 6b. All three rules are applicable to the initial shape. If rule 3 is aplied to the initial shape, the markers are erased, the generation halts, and a shape in the language has been generated. If rule 1 is applied to the initial shape, six terminals are added, the markers are moved, but from that point on only rule 1 is applicable. If rule 1 is applied to the initial shape, the generation continues indefinitely and no shape in the language can ever be produced as rule 3 can never be applied and the markers can never be erased. If rule 2 is applied to the initial shape (as in Figure 6b), nine terminals are added, the markers are moved, and a new hexagon is begun. Rule 1 is then applied until the markers meet and the process is repeated. Note that the size of the markers remains constant throughout the generation. Each shape in L(PSG5) is a reversible figure.

A similar language of reversible figures could be defined using a shape

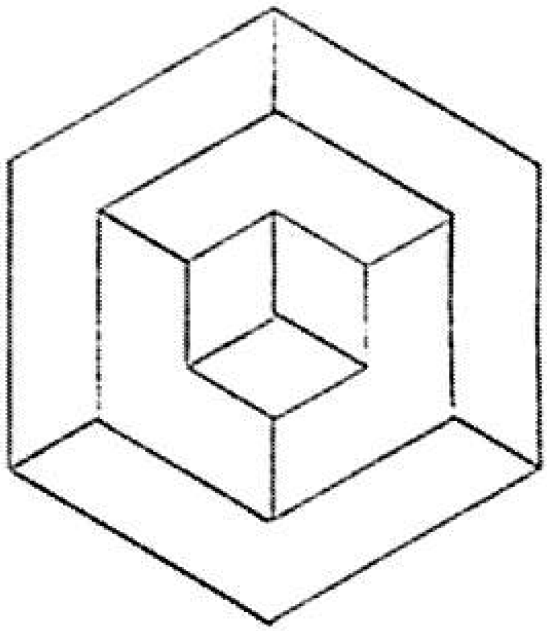


Figure 5a. A new reversible figure.

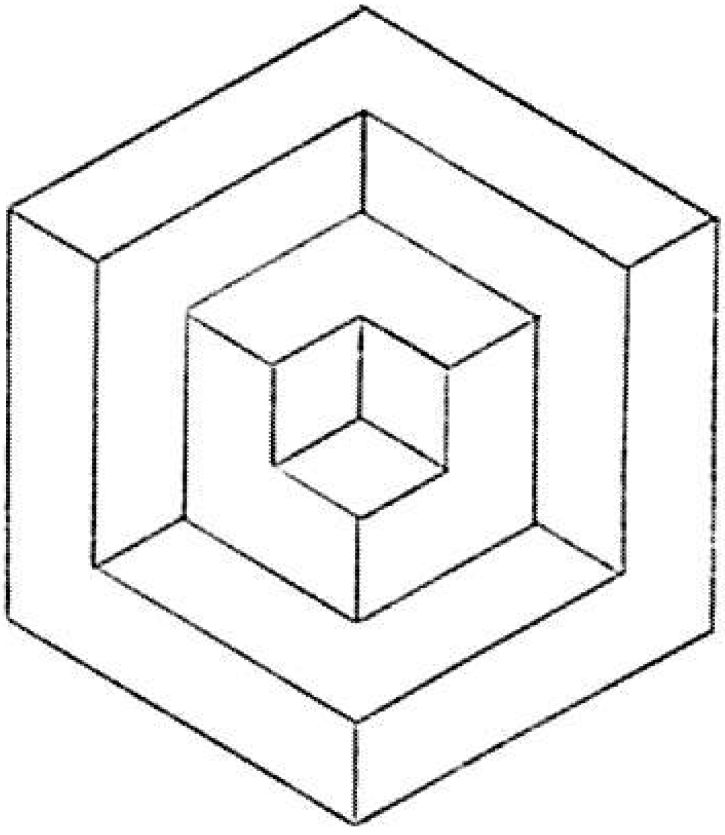
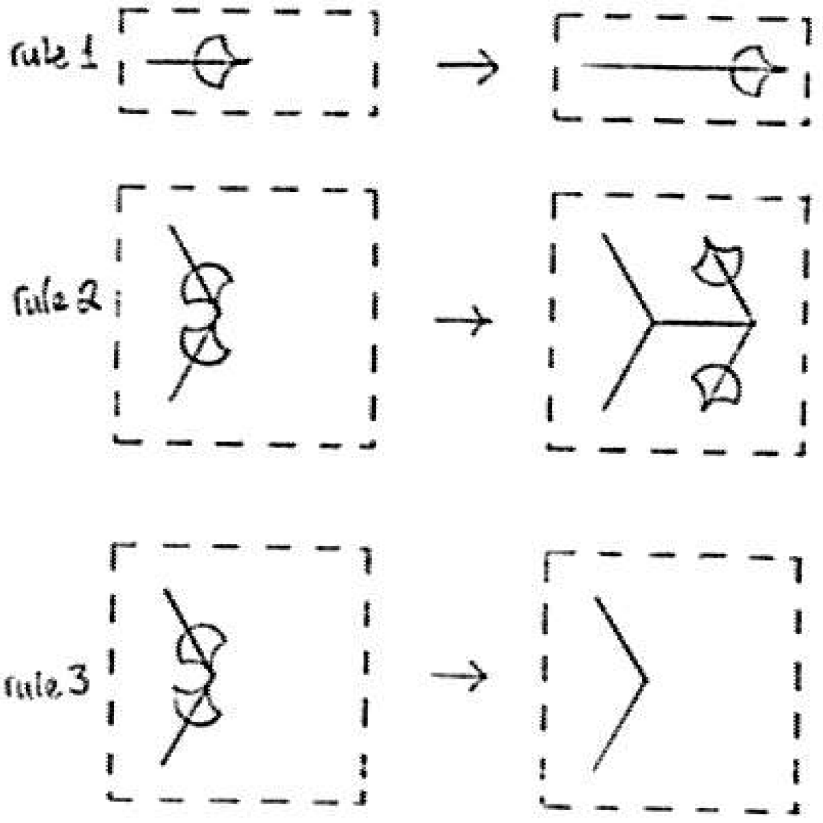


Figure 5b. Variation.

$PSGS = \langle V_T, V_M, R, I \rangle$

$V_T = \{$ — $\}$ $\quad V_M = \{$  $\}$

R contains



I is

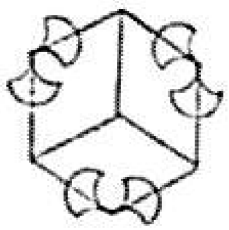


Figure 6a. PSGS, a parallel shape grammar that generates a language of reversible figures.

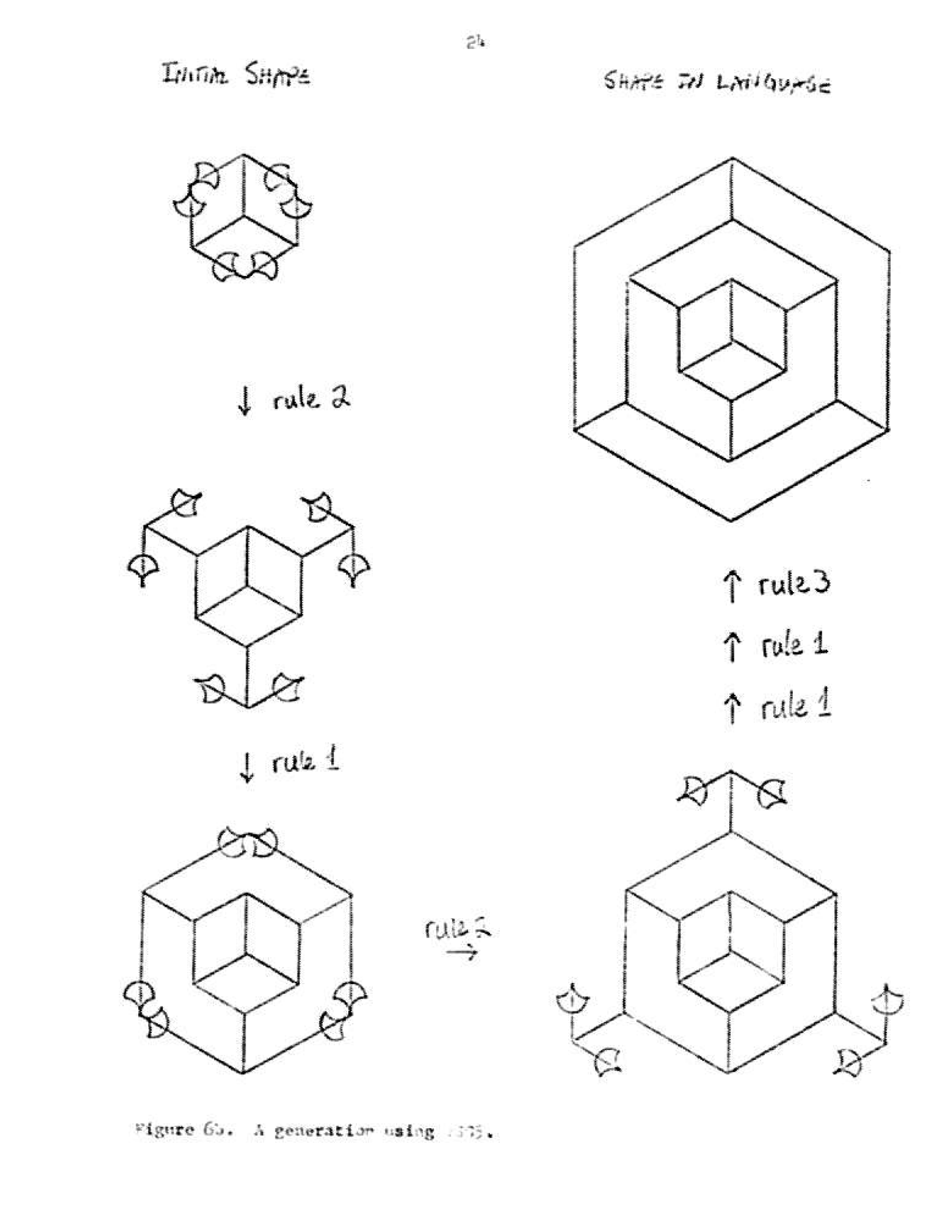


Figure 6b. A generation using [illegible].

grammar of the format of SG2. In each shape of this language, the distances between hexagons would increase geometrically rather than remaining constant as in L(PSG5).

As a short digression, it is interesting to analyze these reversible figures in terms of a contemporary computer vision algorithm. In particular, how does Huffman's algorithm for interpreting two-dimensional figures as three-dimensional objects [Huffman 1971], [Duda and Hart 1973] interpret these reversible figures? Are the figures reversible (ambiguous) for this algorithm? Implicit in Huffman's algorithm is that all objects have discernible width. Thus for this algorithm the figures are not ambiguous. Only one interpretation of Figure 5a is possible and only one interpretation of Figure 5b is possible. But the two interpretations are different! Except for the outermost lines, all lines that are interpreted as convex in Figure 5a are interpreted as concave in Figure 5b and vice versa. The Huffman labelling of Figures 5a and 5b are given in Figure 7a and 7b. Following Huffman, a "+" denotes a line interpreted as a convex edge of the three-dimensional object, a "-" denotes a concave edge, and an "→" denotes a convex occluding edge whose associated visible surface is to the right as one looks along the arrow. Because for the algorithm all objects have width, the convexity or concavity of the central lines of these figures is determined by the number of surrounding hexagons (i.e., 1 plus the number of times rule 2 was applied in the generation of the shape). If the number of hexagons is odd, as in Figure 5a, the three central lines are interpreted as convex by the algorithm. If the number of hexagons is even, as in Figure 5b, the three central lines are interpreted as concave using Huffman's algorithm.

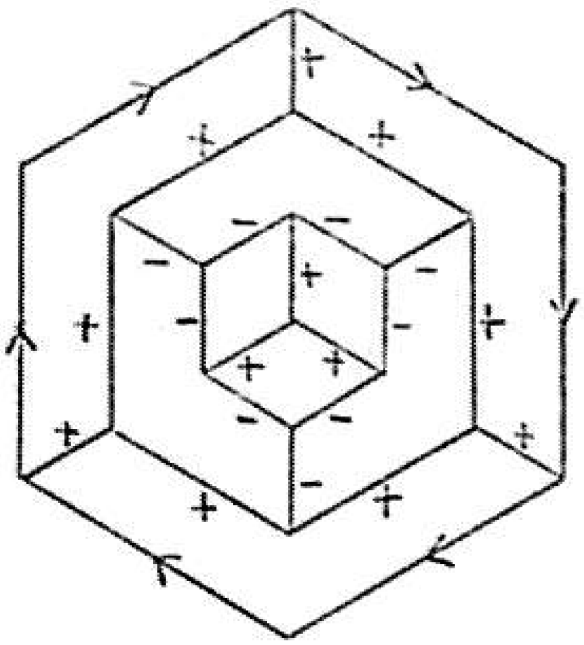


Figure 7a. Huffman labelling for reversible figure.

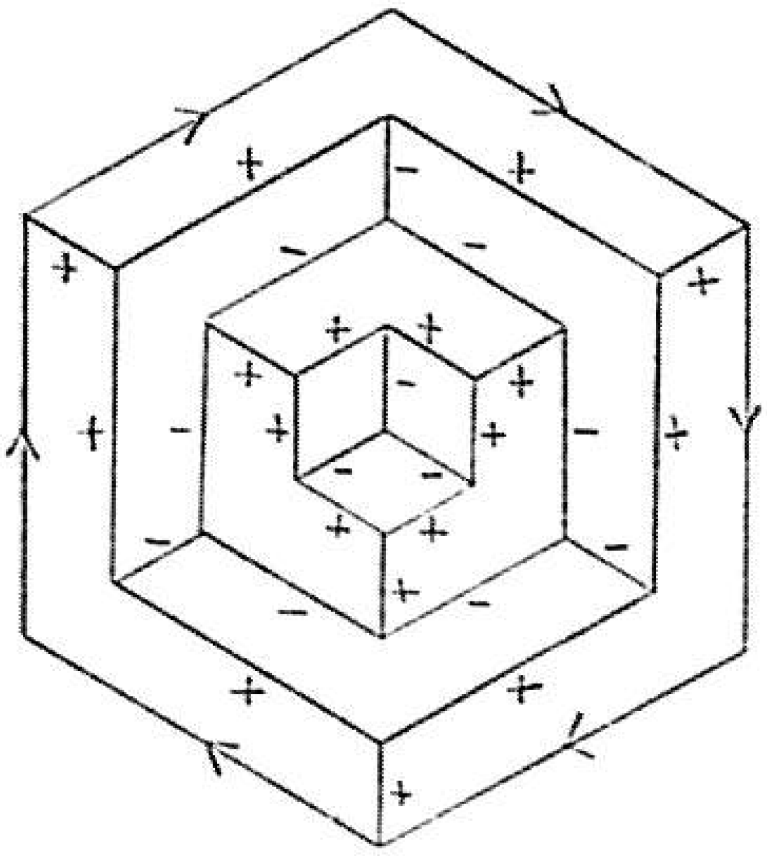


Figure 7b. Huffman labelling for variation.

1.6 Examples: mathematical curves

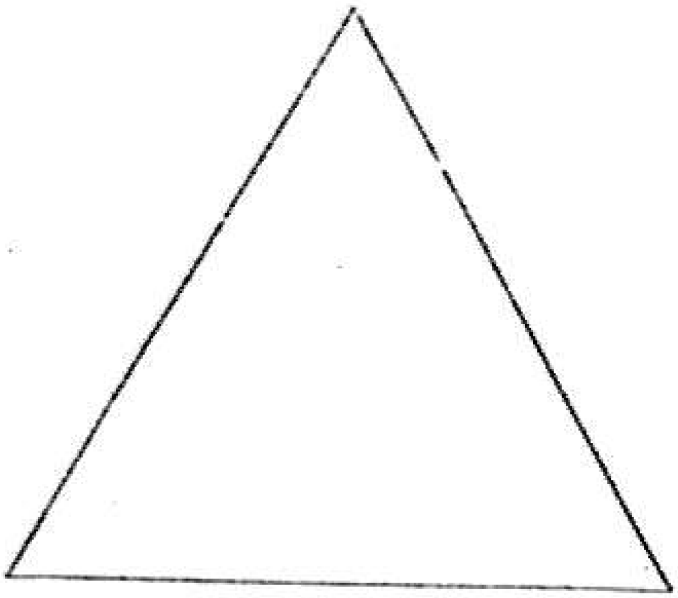
Shape grammars can be used to define a number of classical mathematical curves. Previously, these curves were defined either analytically or by displaying instances of the curves and giving informal English descriptions. Shape grammars provide a method for the precise, algorithmic specification of these curves that at the same time yields insights about the geometrical structure of the curves.

1.6.1 Snowflake curve

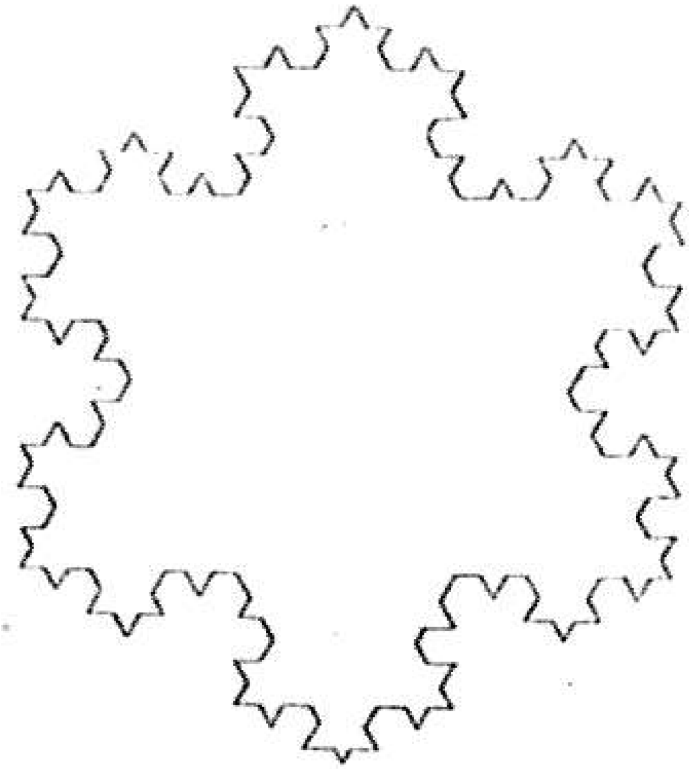
The first four stages, S_0 - S_3, of the Snowflake curve [Kasner and Newman 1965] are shown in Figure 8. The Snowflake curve is interesting because in the limit, the area enclosed by the curve is finite while the length of the curve is infinite. (In the limit, the area of the curve is 1.6 times the area of the original triangle [Kasner and Newman 1965]. At each successive stage, the length of the curve increases by a factor of 4/3. Clearly, $(4/3)^n$ does not converge as n increases.)

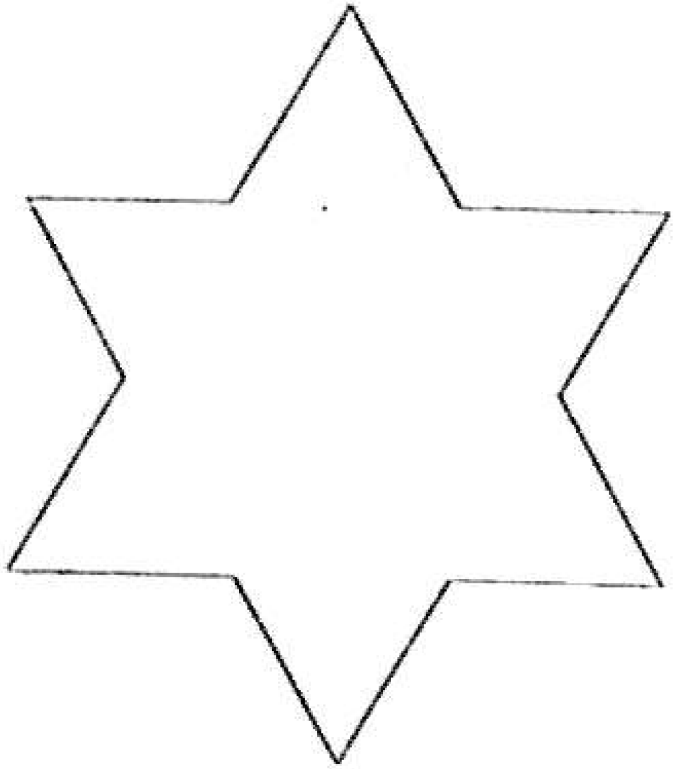
A parallel shape grammar, PSG6, for the Snowflake curve is shown in Figure 9a. Note that the right side of the first shape rule contains four markers and that the initial shape contains three markers. The generation of a shape using PSG9 is shown in Figure 9b. Rules 1 and 2 are applicable under identical circumstances. They are applicable at three different places in the initial shape, at twelve

CURVE 0

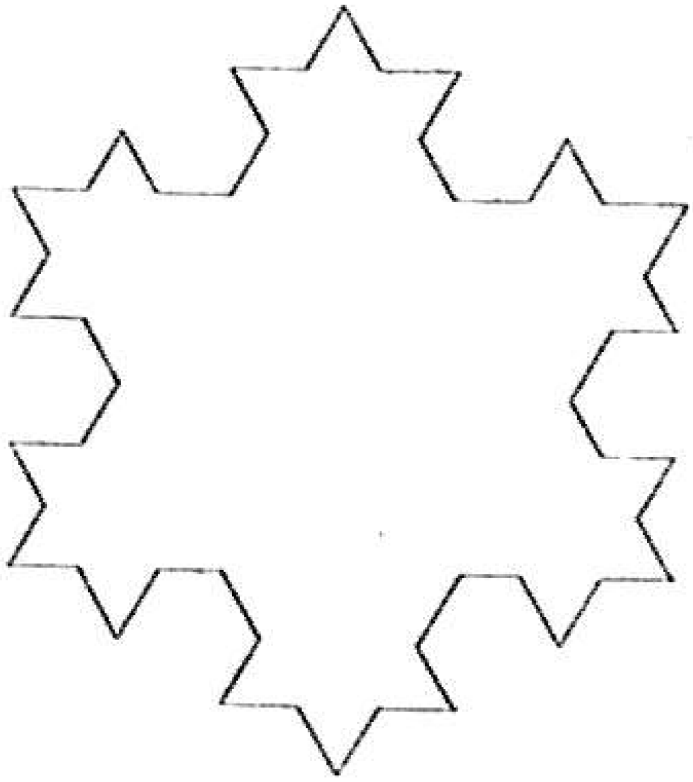


CURVE 3





CURVE 1



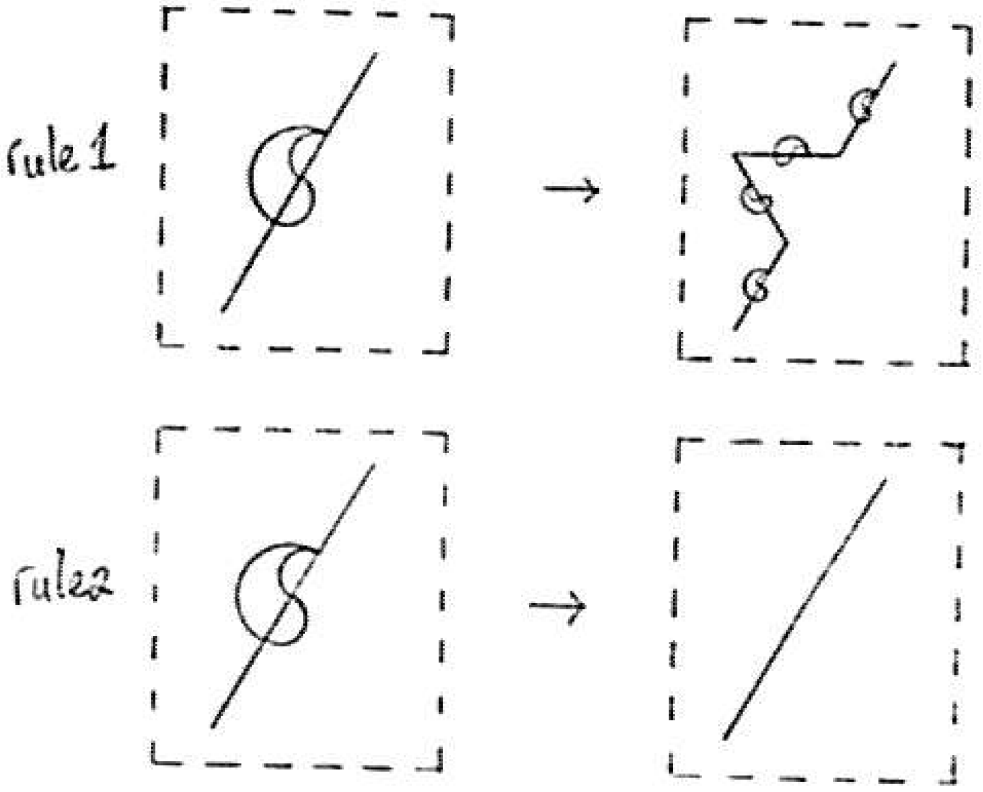
CURVE 2

Figure 8. The first four curves of the Koch snowflake.

$PSG6 = \langle V_T, V_M, R, I \rangle$

$V_T = \{ — \}$ $V_M = \{ \text{S} \}$

R contains



I_{15}

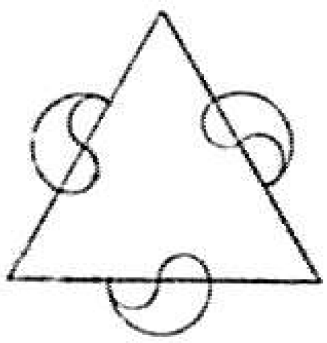


Figure 9a. PSG6, a parallel shape grammar for the Snowflake curve.

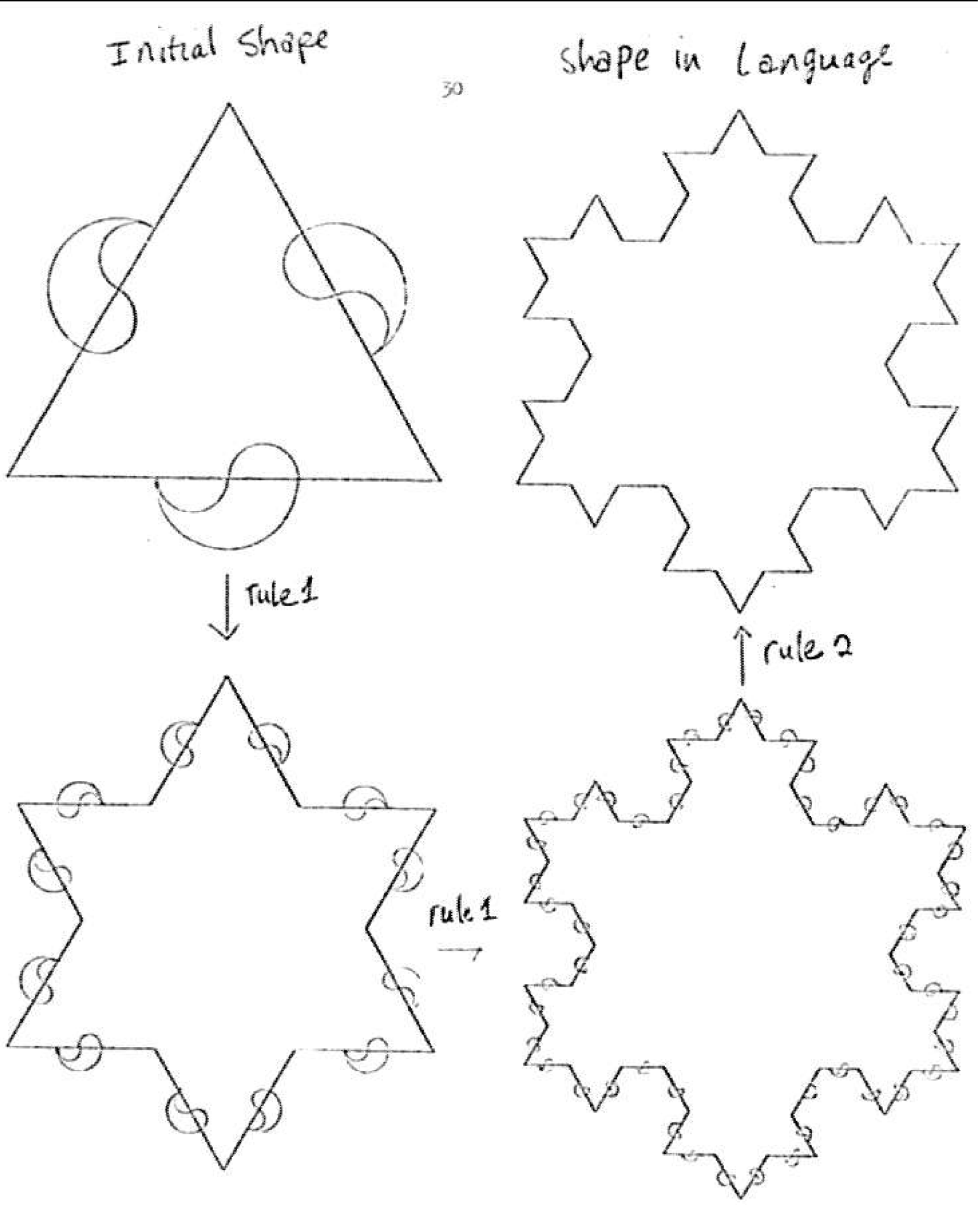


Figure 9. A generation using SG.

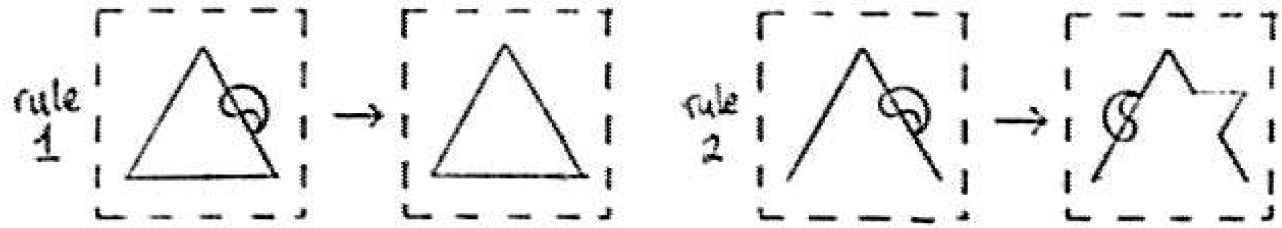
different places in the next shape, etc. Application of rule 1 results in the generation of the next stage of the Snowflake curve and the continuation of the generation process; application of rule 2 halts the generation process. The language generated by PSG9 contains exactly the successive stages of the Snowflake curve.

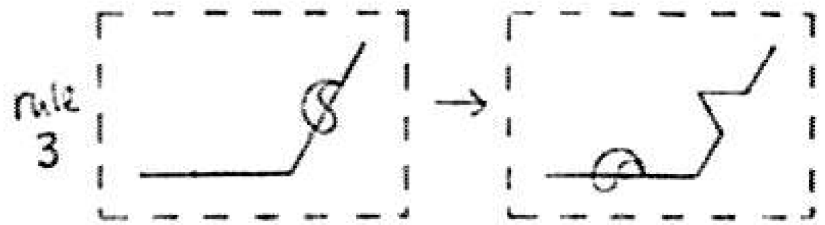
As with SG4, if the shape grammar PSG9 is used in serial, the generation process can proceed to different depths in different parts of the shape. For a (serial) shape grammar to define a language containing only completed stages of the Snowflake curve, it cannot allow the generation process to proceed independently in different parts of the shape. The generation process must be controlled to generate the shape uniformly. A (serial) shape grammar, SG7, that generates just the infinite series of curves S_0, S_1, ... is shown in Figure 10a. The generation of the curve S_2 using SG7 is shown in Figure 10b. The strategy implicit in SG7 is to trace around the shape (using rules 2 and 3), expanding lines as the trace proceeds. The asymmetry of the marker forces the generation to always proceed counter-clockwise around the shape. Whenever a complete trace is made, the generation can either be halted (by applying rule 5) or allowed to proceed (by applying rule 4) for at least another complete trace. Rule 1 is only applicable to the initial shape. Without rule 1, the language would not include S_0. There may well be (serial) shape grammars that are simpler than SG7 that generate the successive stages of the Snowflake curve.

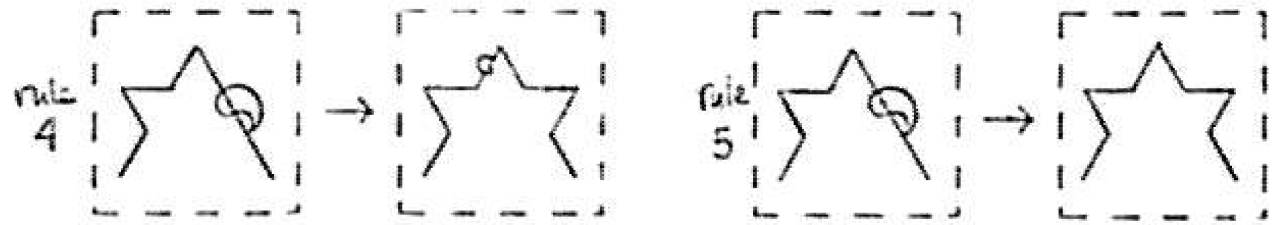
$SGT = \langle V_T, V_M, R, I \rangle$

$V_T = \{ — \}$ $V_M = \{$  $\}$

R Contains







I is

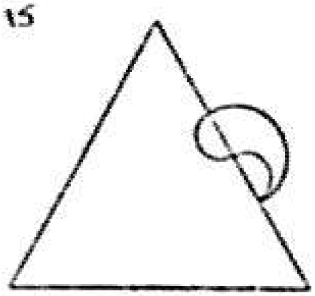


Figure 10a. SGT, a serial shape grammar for the Snowflake curve.

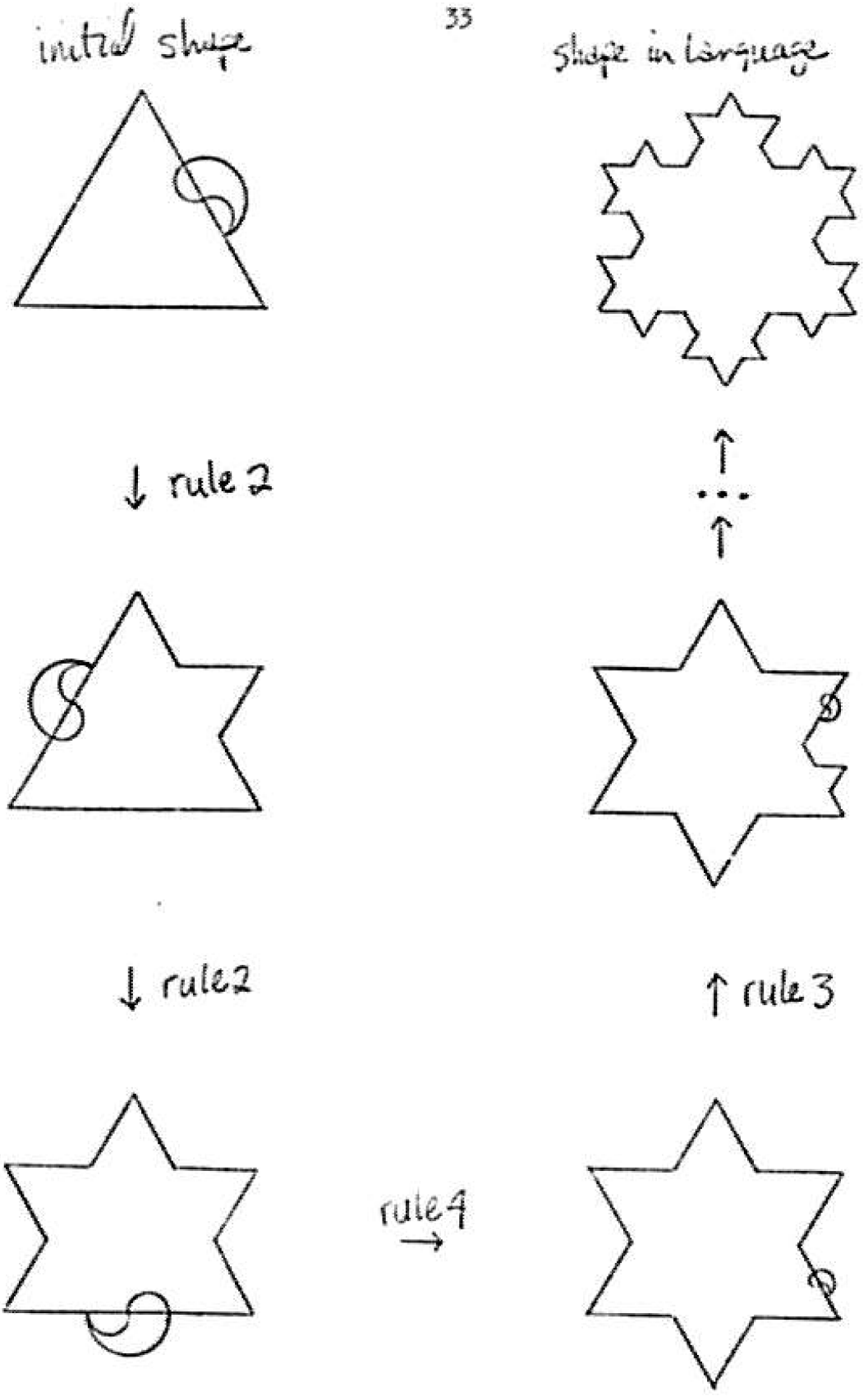


Figure 105. A generation using S67.

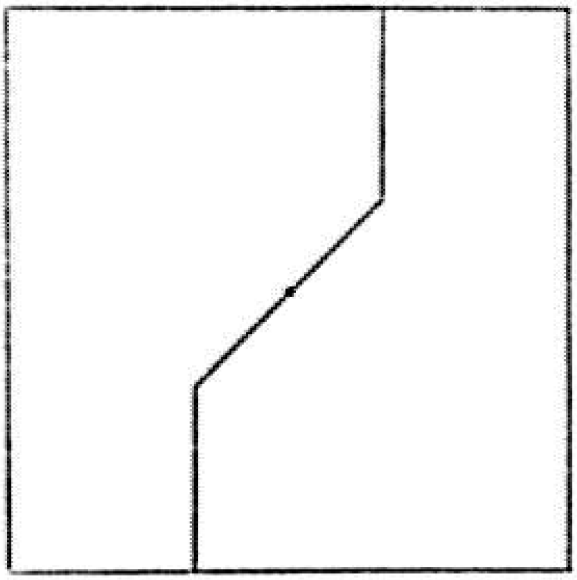
1.6.2 Peano's curve variation

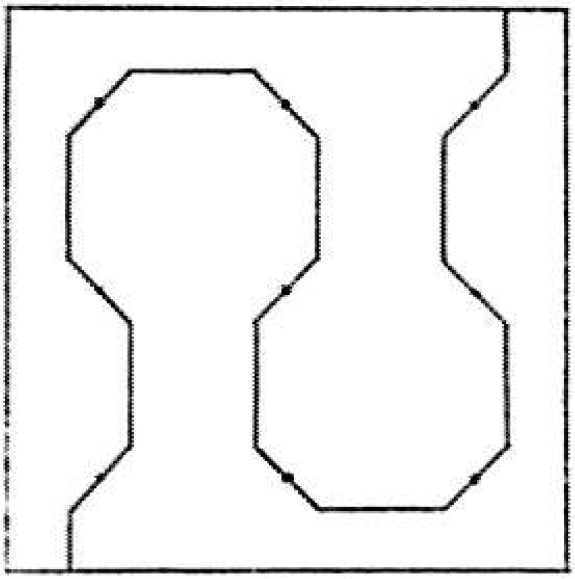
Peano's curve [Peano 1890] is a curve that passes through every point of the unit square. Peano defined the curve analytically, roughly in terms of a parameter t that varies from 0 to 1 and continuous functions f(t) and g(t) defined such that for every (x,y) where $0 < x,y < 1$ there exists a t with f(t)=x and g(t)=y. Moore [1900] represented Peano's curve geometrically as the limit of a series of curves made up of polygonal arcs. The first curve of the series passes through the center of the unit square. Next, the unit square is subdivided into nine equal squares; the second curve of the series passes through the center of each of these subsquares. The third curve passes through the centers of each of the 81 subsquares of the unit square, etc.

A new variation on Peano's curve is illustrated in Figure 11. The first three polygonal curves (P_0, P_1, P_2) of the series are shown with the unit square. The centers of the subsquares that the curves pass through are marked with dots. This curve differs from Peano's curve in terms of the order that the curves pass through the centers of the subsquares.

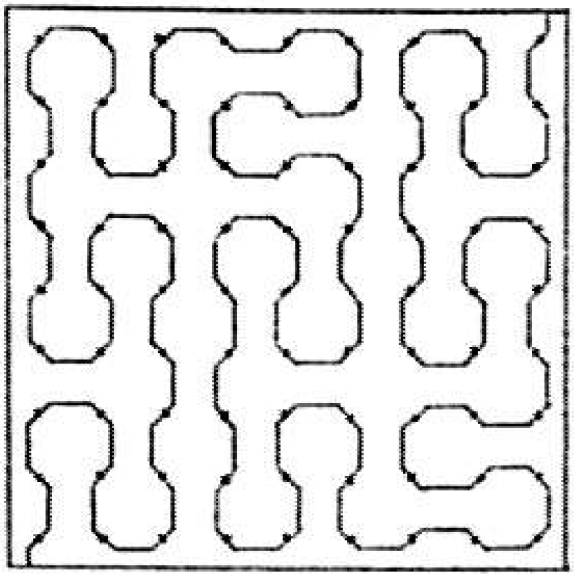
A shape grammar, SG8, that generates exactly this series of curves is shown in Figure 12. As with the generation of the Snowflake curves using SG7, the generation, using SG8, of the curve P_n involves the succesive generation of the curves P_1, P_2, ... P_{n-1}. The generation of curve P_{i+1} from curve P_i proceeds by expanding successive sections of the curve P_i using rules 1 and 2. Examination of the curves reveals that each section of a curve that passes through a subsquare is

CURVE 0





CURVE 1



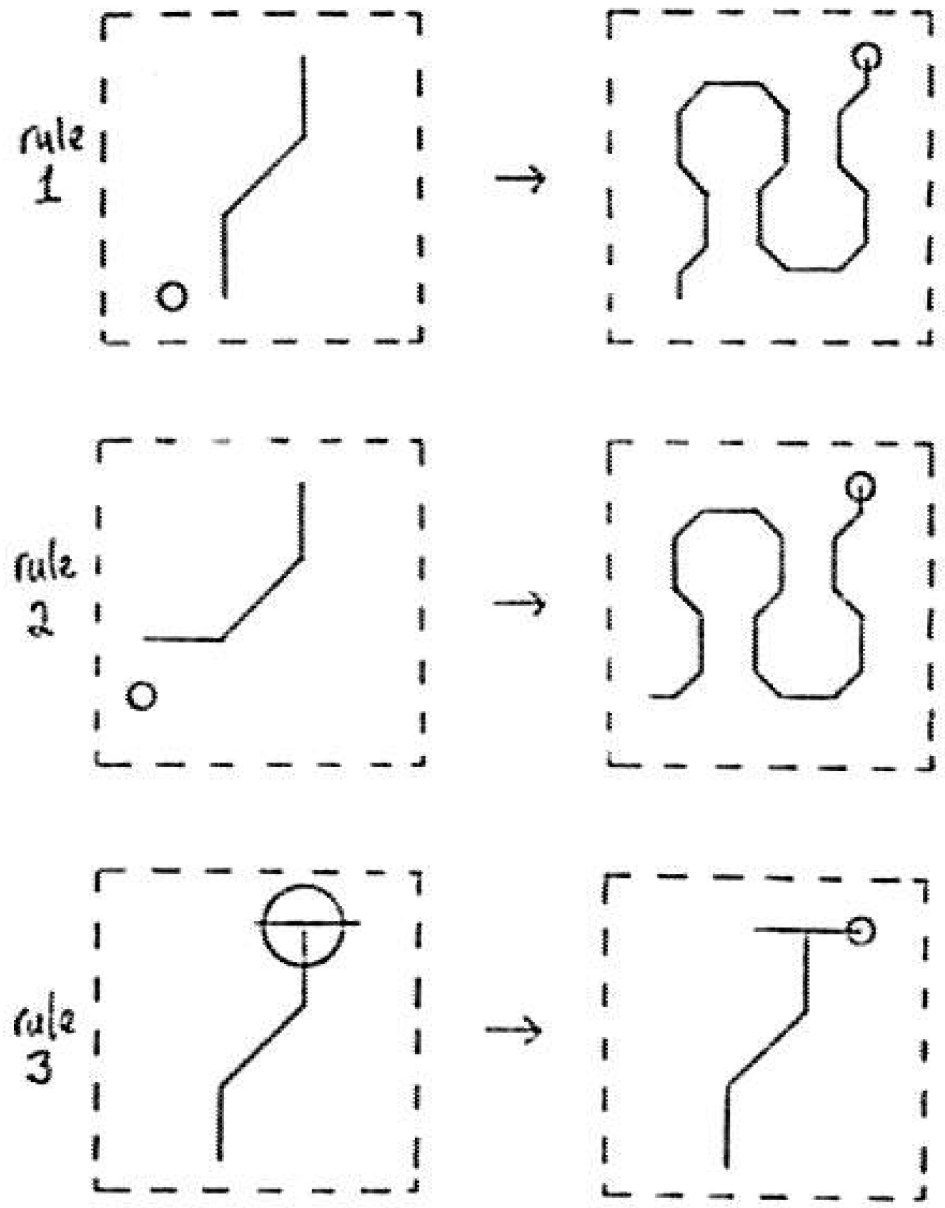
CURVE 2

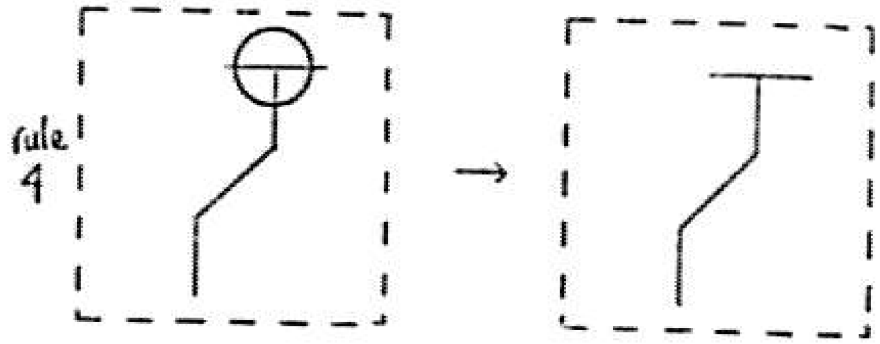
Figure 11. The first three curves of a variation of Peano's curve. (Centers of subsquares are marked with dots.)

SGB = $\langle V_T, V_M, R, I \rangle$

$V_T = \{ — \}$ $V_M = \{ ○ \}$

R contains





I is

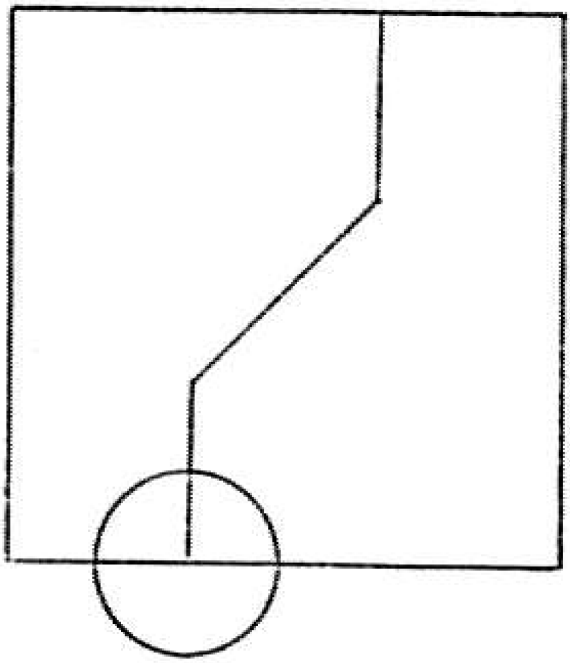


Figure 12. SGI, a shape grammar that generates the Peano's curve variation.

identical to either the terminals in the left side of rule 1 or the terminals in the left side of rule 2 (or their mirror images). The effect of applying either rule 1 or rule 2 is to replace the section of a curve that passes through (the center of) a square with a curve that passes through (the centers of) the nine subsquares. That is, each application of rule 1 or rule 2 replaces a section of curve P_i with the corresponding section of curve P_{i+1}. The center of the marker in the left sides of rules 1 and 2 shows the exact location of the beginning of the terminals added in the right sides of the rules. When the last section of a curve is reached i.e., when the marker reaches the the edge of the unit square, the generation is either halted (by applying rule 4) or forced to continue (by applying rule 3) for a complete trace back along the just generated curve.

1.6.3 Hilbert's curve

Hilbert's curve [Moore 1900] is the best known space-filling curve and has appeared in the popular literature of both mathematics [Hahn 1954] and art [Munari 1965], frequently labelled erroneously as Peano's curve. The sequence of curves, H_i, used in the definition of Hilbert's curve is similar to the sequence P_i of Peano's curve, but is generated by recursively subdividing the unit square into four subsquares rather than nine. Curves H_0, H_1, H_2, and H_5 are shown in Figure 13. H_0 passes through the four subsquares of the unit square, H_1 through the sixteen subsquares, etc.

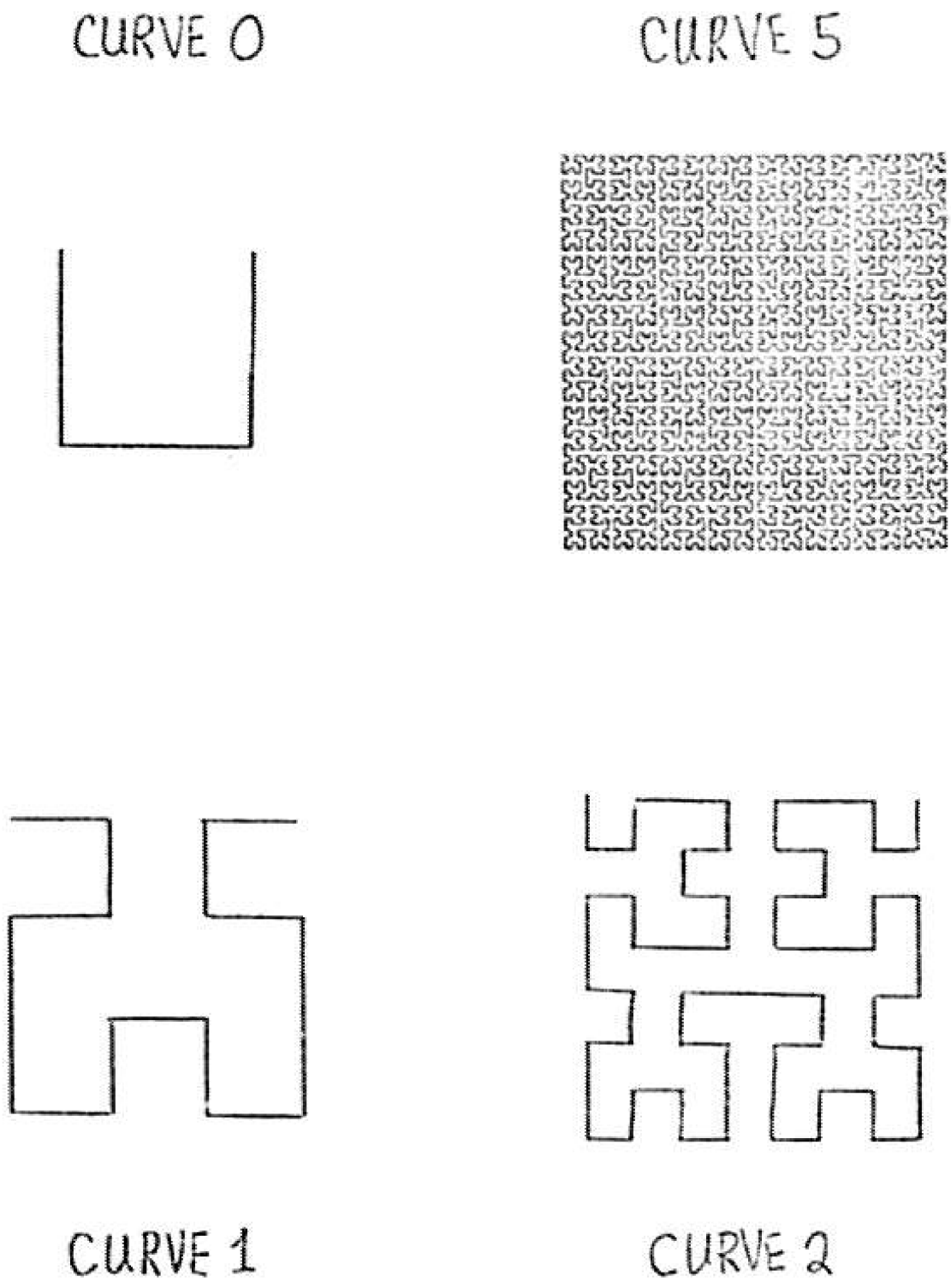


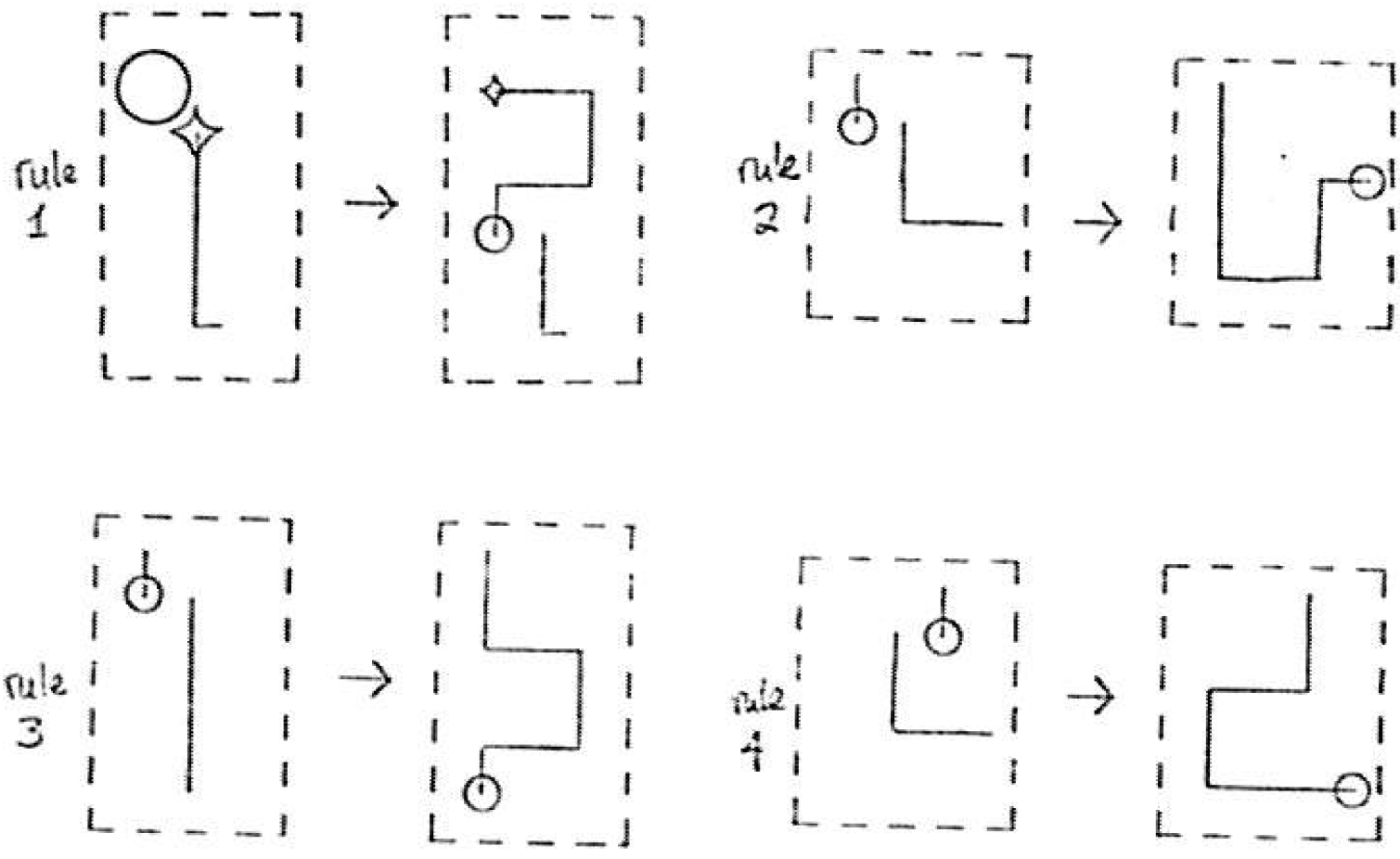
Figure 13. The first three and sixth curves of Hilbert's curve.

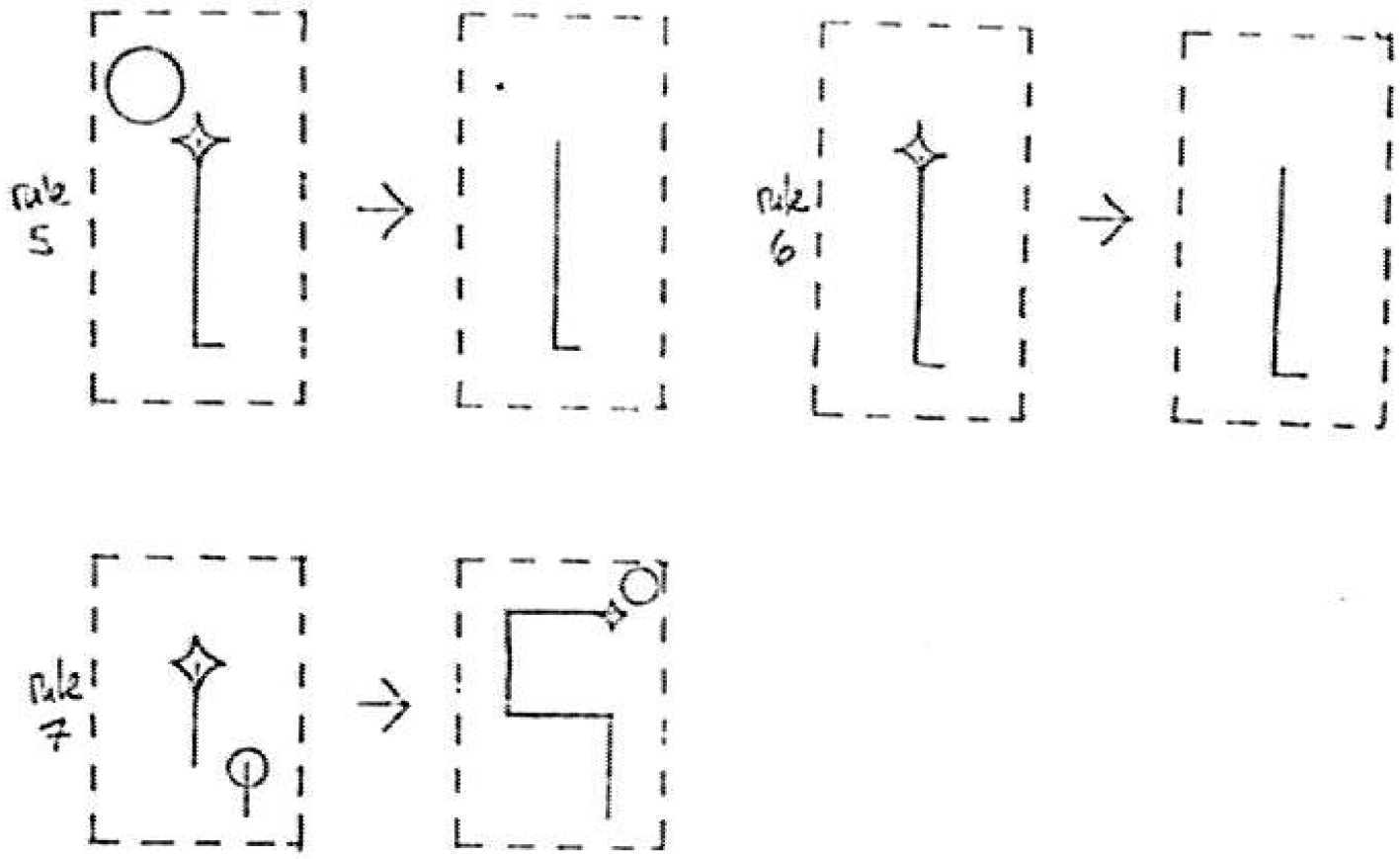
A shape grammar, SG9, that generates just the sequence of curves H_i is shown in Figure 14a. In the generation of curve H_n using SG9, curves H_1 ... H_{n-1} are first generated. The grammar contains two markers, a curved diamond and a circle. The diamond is used to mark the endpoints of the curve during the generation process. This is necessary because the locations of the endpoints of curve H_i are different than the locations of the endpoints of curve H_{i-1} and there are no convenient landmarks. The circle is used to trace around the curves. The grammar contains seven shape rules. Rule 1 is used at the beginning of the generation of each H_i to expand the section of the curve in the initial subsquare. Rules 2 - 4 are the core of the shape grammar; they are used to successively expand the section of the curve contained in all but the initial and final subsquares. Rule 7 is used at the end of the generation of each H_i to expand the section of the curve in the final subsquare. Rule 5 is an alternative to rule 1; application of rule 5 causes the erasure of the circle marker and one of the diamond markers and results in the end of the generation process. Rule 6 is used to erase the diamond marker not erased by rule 5. While rule 6 is applicable at each step in the generation, if it is applied prematurely the generation comes to a dead end as it becomes impossible to apply rule 5 and thereby erase the circle marker. The generation of H_2 using this shape grammar is shown in Figure 14b.

$SGG = \langle V_T, V_M, R, I \rangle$!

$V_T = \{ \text{—} \}$ $V_M = \{ \bigcirc, \diamondsuit \}$

R contains





I is

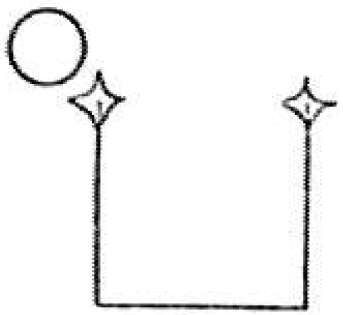
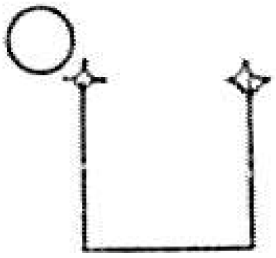
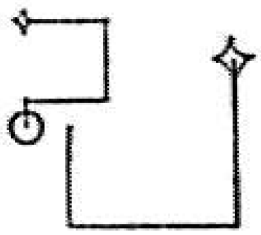


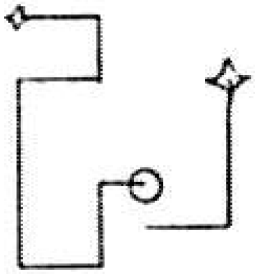
Figure 14a. SCG, a shape grammar that generates Hilbert's curve.



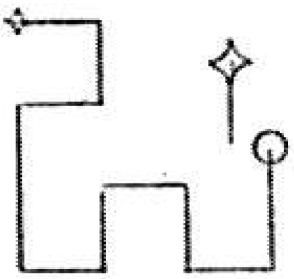
↓ rule 1



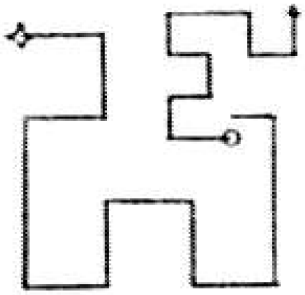
↓ rule 2



↓ rule 4

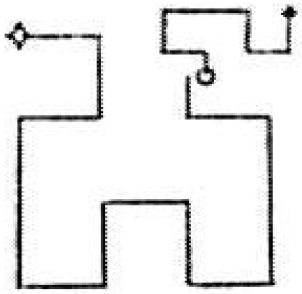


rule 7 →

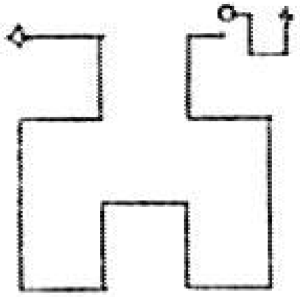


rule 4 →

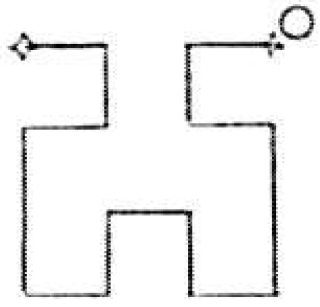
↑ rule 4



↑ rule 2



↑ rule 1



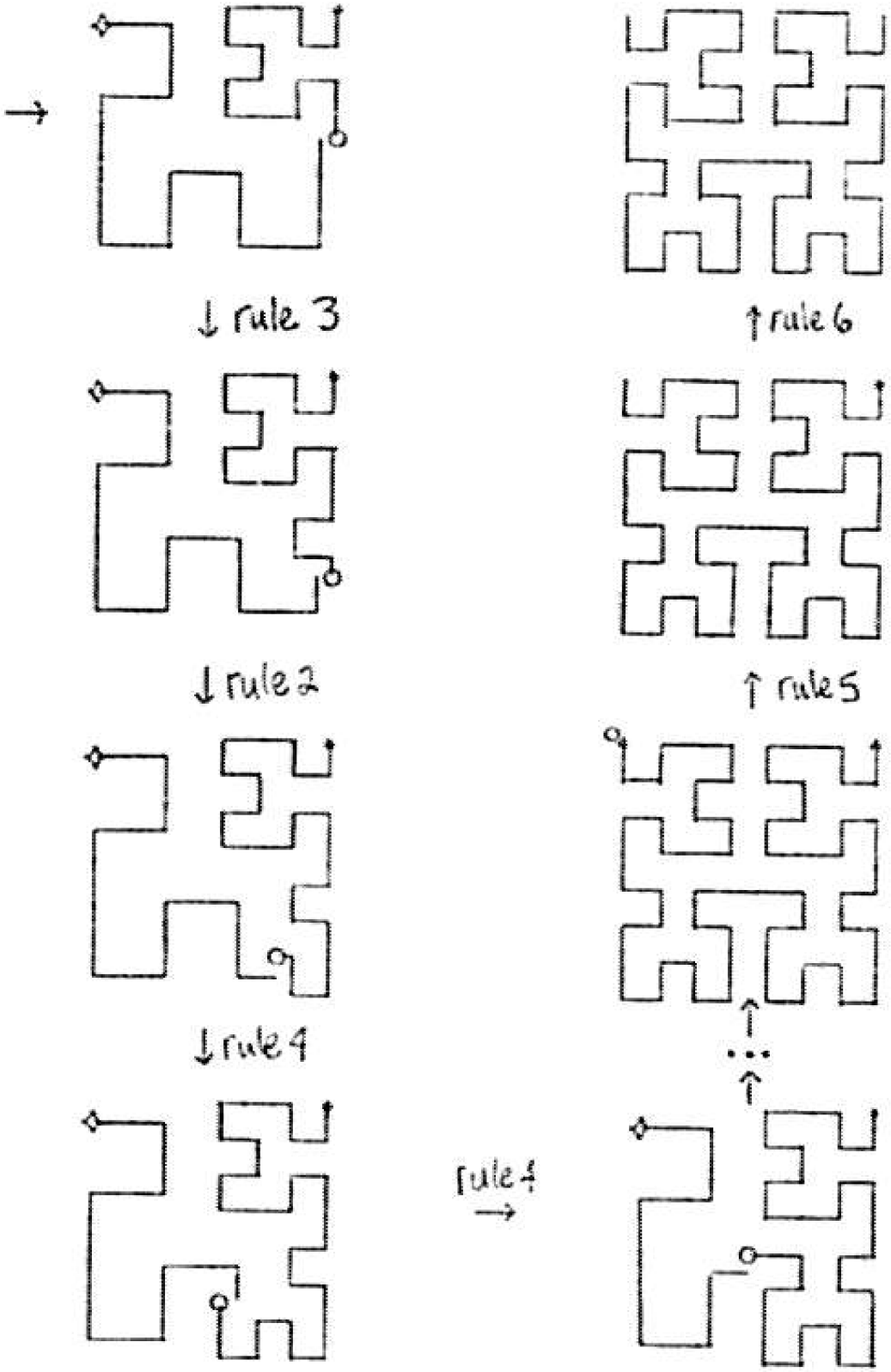


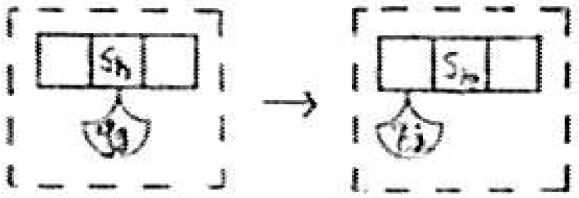
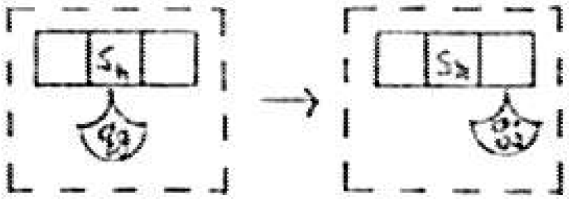
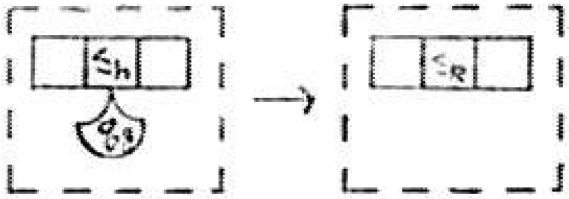
Figure 44b. A generation using 309.

1.7 Simulation of Turing machines

The shape grammars presented so far have generated languages of abstract shapes. It is also possible to use shape grammars for symbol processing by regarding the symbols as shapes. In this section a straight-forward method for constructing a shape grammar for an arbitrary Turing machine is presented.

There are several ways of specifying Turing machines. The method used here is basically that of Minsky [1967]. A Turing machine is defined in terms of a finite set of tape symbols, a finite set of states, a description of the operation of the finite state control and tape head, and an initial configuration. Let the tape symbols be denoted s_0, s_1, s_2, ... s_m where s_0 is the blank symbol. Let the states be denoted by q_1, q_2, ... q_n, halt. The operation of the finite state control and tape head is specified by a list of quintuples of the form (old state, symbol scanned, new state, symbol written, direction of motion of the tape head). Each quintuple is of one of three types: (q_i, s_h, q_j, s_k, left), (q_i, s_h, q_j, s_k, right), or (q_i, s_h, halt, s_k, -). The initial configuration is specified by an input tape, a starting position on the tape for the tape head, and an initial state, q_0. The tape is of arbitrary length but initially it must contain only a finite number of non-blank symbols.

The method for constructing a shape grammar for simulating a Turing machine specified in this form is outlined in Figure 15. V_t contains the tape symbols, a square for forming the Turing machine tape, and a half-square with a jagged edge for indicating the edges of the tape. V_m contains the symbols of the non-halting states of the Turing machine plus a marker that is used to represent the tape head.

TURING MACHINE	EQUIVALENT SHAPE GRAMMAR
FINITE SET OF TAPE SYMBOLS $\{S_0, S_1, \ldots S_M\}$ where S_0 is "blank"	$V_T =$ $\{ \square, \boxed{S}, S_0, S_1, \ldots S_M \}$
FINITE SET OF STATES $\{q_1, q_2, \ldots q_n, \text{halt}\}$	$V_M =$ $\{ \text{⌓}, q_1, q_2, \ldots q_n \}$
FINITE STATE CONTROL There are three types of 5-tuples:	SHAPE RULES Corresponding Shape rules:
$(q_g, S_h, q_j, S_k, \text{left})$	
$(q_g, S_h, q_j, S_k, \text{right})$	
$(q_g, S_h, \text{halt}, S_k, -)$	

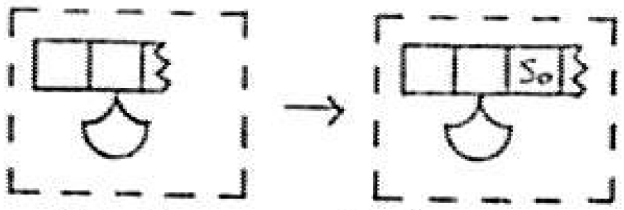
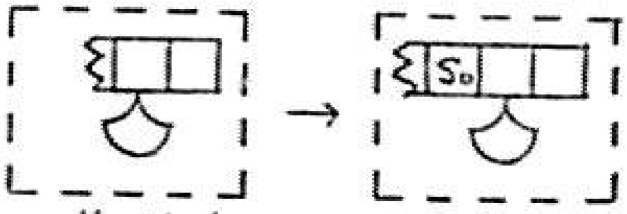
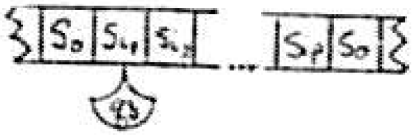
TURING MACHINE	EQUIVALENT SHAPE GRAMMAR
	additional shape rules
	 add a blank square to the right end
	 add a blank square to the left end
INITIAL CONFIGURATION	INITIAL SHAPE
... S_0 S_{i_1} S_{i_2} ... S_{i_p} S_0 ... 	

Figure 15. A method for constructing a shape grammar that simulates a given Turing machine.

As by the definition of shape grammars, $V_t \cap V_m = \phi$, if the same symbol is used to indicate both a state and a tape symbol in the definition of the Turing machine, a new symbol not in $V_t \cup V_m$ must be used for one instance of the symbol in the shape grammar. There is one shape rule for each quintuple in the specification of the Turing machine. Each of these shape rules effectively simulates the effect of the corresponding quintuple. The exact form of the shape rules for each of the three types of quintuples is shown in Figure 15. There are two additional shape rules for expanding the tape when necessary by adding a square with a "blank" to either end (see Figure 15). The initial shape is the non-blank part of the input tape with a blank square and then a tape edge terminal on either end and the tape head marker and initial state marker under the starting square of the tape.

In a generation using a shape grammar of this type, there is one shape rule application for each scanning of a tape symbol by the corresponding Turing machine plus whatever shape rule applications are necessary to expand the tape during the generation. The shape generated by a shape grammar constructed in this manner is the output tape of the corresponding Turing machine.

The construction method presented works for both deterministic and non-deterministic Turing machines. If the Turing machine is deterministic and eventually halts given the input tape, the language defined by the corresponding shape grammar contains one shape. If the Turing machine is non-deterministic, the language of the corrsponding shape grammar may contain multiple shapes, i.e., one for each of the possible output tapes of the Turing machine.

This example shows the general computing power of unrestricted shape grammars.

1.8 Related work

Since the early 1960's there has been a considerable amount of work on developing grammars that define languages of pictures rather than strings. Much of this research has been directed toward finding formalisms that are useful for the automated analysis of various types of pictures. This work is usually included under the heading "syntactic pattern recognition" or "linguistic pattern recognition". Reviews of early work in this field can be found in [Miller and Shaw 1968] and [Fu and Swain 1971]. A short review of more recent papers is included in [Rosenfeld 1973a].

There seem to be as many varieties of picture grammars as there are papers on the subject. (Indeed, this paper upholds that tradition.) It is difficult to classify all the formalisms, but most seem to fall under one of four general categories.

1.8.1 Array grammars

An array grammar defines a language of n-dimensional (n usually is 2) arrays of symbols. The earliest array grammar, a grammar that generates a language of isosceles right triangles, was developed by Kirsch [1964]. A formal definition of array grammars is given in [Milgram and Rosenfeld 1972]. Normal forms and a number of interesting formal properties of array grammars appear in [Rosenfeld 1973b].

$AG = \langle \{S, T\}, \{1\}, \#, P, S \rangle$

where the productions P are

(1) $\begin{matrix} \# \\ S\ \# \end{matrix} \rightarrow \begin{matrix} S \\ 1\ T \end{matrix}$ (2) $S \rightarrow 1$

(3) $\begin{matrix} T \\ \# \end{matrix} \rightarrow \begin{matrix} 1 \\ \# \end{matrix}$ (4) $\begin{matrix} T \\ 1\ \# \end{matrix} \rightarrow \begin{matrix} 1 \\ 1\ T \end{matrix}$

Figure 16a. An array grammar that generates a language of isosceles right triangles (from [Mercer and Rosenfeld 1973]).

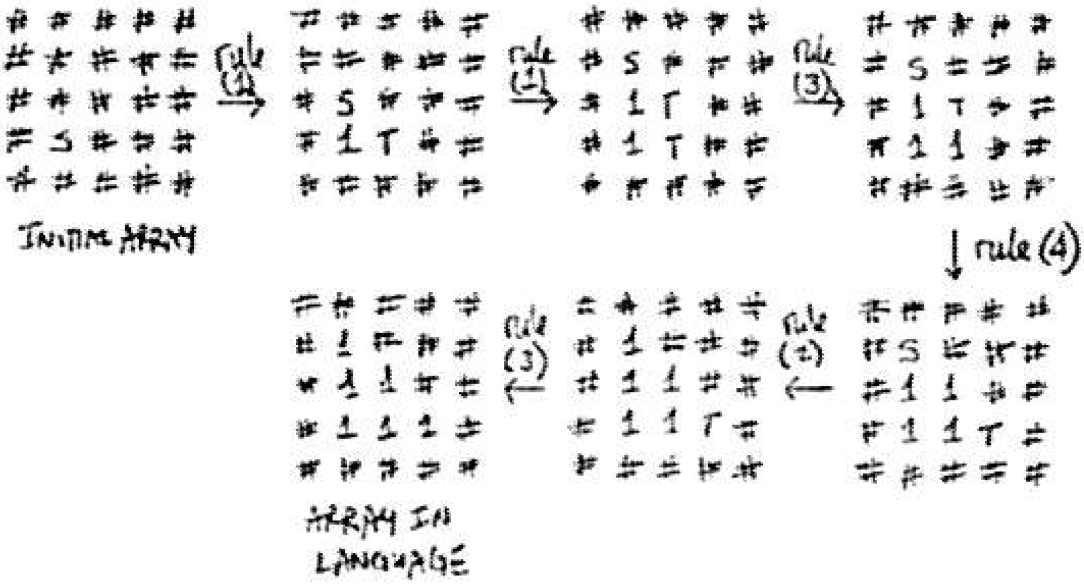


Figure 16b. A generation using the array grammar of 16a.

$PAG = \langle \{S\}, \{1\}, \#, P, S \rangle$

where the productions P are

(1) $\begin{matrix} \# \\ S\ \# \end{matrix} \rightarrow \begin{matrix} S \\ 1\ S \end{matrix}$ (2) $S \rightarrow 1$

Figure 16c. A parallel array grammar that generates the identical language of isosceles right triangles (from [Mercer and Rosenfeld 1973]).

An array grammar that generates a language of isosceles right triangles is shown in Figure 16a. This grammar, which is simpler than Kirsch's, is taken from [Mercer and Rosenfeld 1973]. An array grammar is specified by a quintuple where the first element is the set of non-terminal symbols, the second element is the set of terminal symbols, the third is the blank symbol, the fourth is a set of productions, and the fifth is the initial symbol. The initial array consists of a single instance of the initial symbol surrounded by a field of blank symbols. Productions are rules for substituting sub-arrays. If during the generation process the array contains a sub-array identical to the left side of a rule, the right side of the rule may be substituted for that sub-array. A derivation using this array grammar is shown in Figure 16b. This array grammar is an example of an isotonic array grammar [Rosenfeld 1971], which is an array grammar in which the left and right sides of a production refer to identical elements of the array. Parallel array grammars are array grammars in which every instance of the left side of a rule is replaced by the right side, rather than just one instance. A parallel array grammar that defines the same language as does the (serial) array grammar of Figure 16a is shown in Figure 16c. This grammar is also from [Mercer and Rosenfeld 1973].

There is a substantial body of literature related to array grammars. Array grammars have been used to specify languages of polygons [Dacey 1970], [Dacey 1971] and crystallographic patterns [Siromoney et al. 1973]. Conway's popular "game of life" [Gardner 1970] can be defined using parallel array grammars as can related work on recursively defined growth patterns [Schrandt and Ulam 1967]. For picture processing, parallel array grammars can be used to define most local

operators for edge-finding, noise elimination and skeletonization. There is a package of Fortran procedures for simulating array grammars [Mercer and Rosenfeld 1973]. Turing machines defined on multidimensional tapes [Blum and Hewitt 1967] have been related to array grammars [Milgram and Rosenfeld 1972], as have cellular automata [Smith 1971] and Markov algorithms [Maggiolo-Schettini 1973].

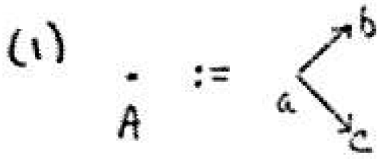
1.8.2 Graph grammars

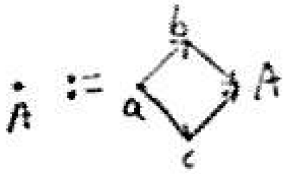
There are several varieties of grammars that generate different types of graphs. Perhaps the best known of these formalisms is the web grammar [Pfaltz and Rosenfeld 1969]. A web is a labelled directed graph, i.e., a directed graph where each node is labelled and/or each edge is labelled [Pfaltz 1972]. Web grammars generate languages of webs. A very simple (node-labelled) web grammar is shown in Figure 17a. This example is taken from [Pfaltz and Rosenfeld 1969]. A web grammar is defined by a vocabulary of non-terminals, a vocabulary of terminals, an initial web, and a set of web rewriting rules. A web rewriting rule is a triple where the first two elements are webs and the third element is an "embedding" which specifies how to substitute the second element for the first element when the rule is applied. The embedding in the example, $E = \{(p,a)|(p,A)\}$, states that if a node, "p", is connected to node "A" in the host web before rule application to node "A", node "p" is connected to the newly added node "a" after

$WG = < V_N, V_T, R, I >$

$V_N = \{ A \}$ $\quad V_T = \{ a, b, c \}$

The rewriting rules, R are

(1) 

(2) 

where for both rules $E = \{ (P, a) \mid (P, A)$ an edge in the host web $\}$

I consists of 

Figure 17a. A simple web grammar (from [Pfaltz and Rosenfeld 1969]).

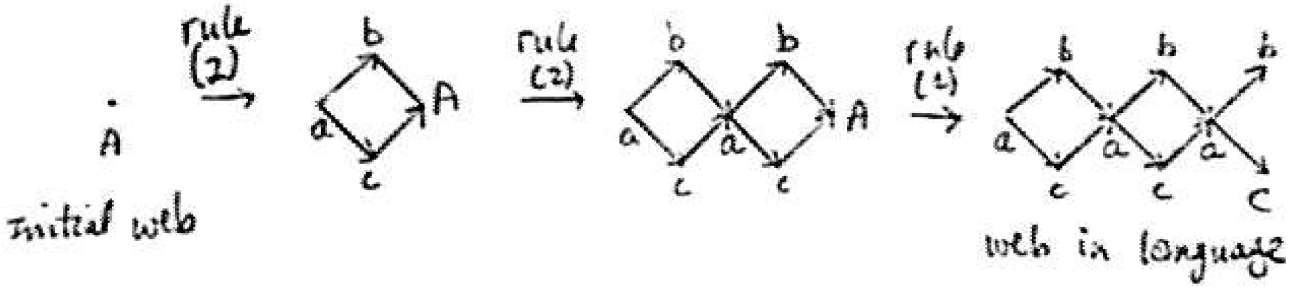


Figure 17b. A generation using the web grammar.

rule application. An example of the generation of a web using this web grammar is given in Figure 17b.

Web grammars have been related to various types of graphs [Montanari 1970] and maps [Rosenfeld and Strong 1971]. Context-sensitive and context-free web grammars [Pfaltz and Rosenfeld 1969] and parallel web grammars [Pfaltz 1972] can be defined. In [Rosenfeld and Milgram 1972], normal forms for web grammars are developed and equivalences are established between classes of web grammars and classes of automata with graph-structured tapes. In [Pfaltz 1972] the utility of web grammars in picture processing is discussed in terms of using webs to represent properties and relationships of elements of pictures and a program for parsing pictures of "neural networks" is described.

Pavlidis [1972] has defined a formalism for graph grammars that is similar to web grammars but that uses a different method for determining embeddings in rule application. Schwebel [1972] has implemented a computer language for graph-structure transformations. Mylopoulos [1972] has investigated the relation between graph grammars and graph automata.

A plex grammar [Feder 1971] can be considered a variety of graph grammar. A plex grammar generates a language of plex structures, which are structures of "n-attaching point entities" (or "NAPEs"). A NAPE is a symbol that has an arbitrary number of points where it can be attached to other symbols. Plex grammars are part of a large body of work in syntactic pattern recognition in which pictures are described in terms of primitive elements that are attached to each other at certain places. This approach has been used by Eden [1968] for handwriting, Narasimhan

[1966] for English letters, Ledley [Ledley et. al. 1965] for chromosome outlines, and Shaw [1969, 1970] for spark chamber pictures, among many, and is reviewed in [Uhr 1971].

Shaw [1972] gives a good general review of graph grammars and their use in picture processing.

Tree grammars are grammars that generate languages of various types of trees. Brainerd [1969] has investigated tree grammars and their relation to tree automata. The use of tree grammars in pattern recognition is discussed in [Fu and Bhargava 1972] and [Williams 1973].

1.8.3 Picture description grammars

There is a substantial body of work on using grammars to specify pictures in terms of descriptions of the pictures. These grammars are usually called pattern grammars or picture description grammars. Frequently the picture descriptions are made in terms of primitive elements of the pictures (such as line segments or circles) and predicates or relations (such as "inside", "above", or "larger than") defined on those elements.

A very simple example of a picture description grammar is shown in Figure 18a. The grammar specifies a class of primitive line-drawings of faces, such as the one in Figure 18b. This example is taken from [Evans 1971]. The primitive picture elements for this grammar are circles, squares, line segments, and dots.

face → (x,y) : features (x), head (y) : inside [x,y]
head → (x) : circle (x)
features → (x,y,z) : eyes (x), nose (y), mouth (z) :
above [x,y] ∧ above [x,z] ∧ above [y,z]
eyes → (x,y) : dot (x), dot (y) : left [x,y]
nose → (x) : square (x)
mouth → (x) : lineseg (x) : horiz [x]

Figure 18a. A simple picture description grammar.

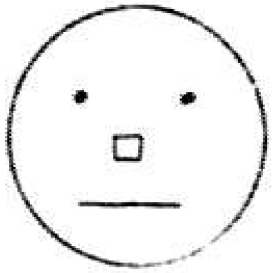


Figure 18b. A face specified by the picture description grammar.

(Figure 18 is from [Evans 1971]).

The predefined predicates are inside[x,y] (meaning object x is inside object y), left[x,y], above[x,y], and horiz[x,y]. x and y are dummy variables. The first rule in the grammar, "face → (x,y): features(x), head(y): inside[x,y]" can be read as: A "face" is an object with two constituents (called x and y) where x is of object type "features" and y is of object type "head" and x is inside of y. This grammar is extremely simple; normally the rules are recursive.

Evans [1969] has written a program that accepts a grammar such as the one in Figure 18a as input and analyzes an input pattern in terms of the grammar by producing "structural descriptions" of the pattern. Evans [1971] has also written a program that infers a grammar which uses a fixed set of primitives for a given set of input patterns. Other formalisms for picture description grammars have been developed by Menninga [1971] who also implemented a program for the top-down analysis of pictures using the grammars, Clowes [1969], and Narasimhan [1970].

The line between picture description grammars and other types of grammars is fuzzy because just about any type of grammar can be written as a picture description grammar given the proper primitives. For example, phrase structure grammars can be encoded using only a single predicate, adj[x,y], which is true if the substring x is immediately left-adjacent to the substring y; the phrase structure production "C → AB" can be encoded "C → (x,y): A(x), B(y): adj[x,y]" [Evans 1971]. A case in point is Shaw's [1970] picture description grammar for spark chamber pictures, which is an encoding of a type of graph grammar which specifies how certain primitive elements can be attached to each other in a picture.

1.8.4 Grammars with coordinates

In grammars with coordinates, terminal and non-terminal symbols have coordinates associated with them. The rewriting rules contain functions which compute the coordinates of the new symbols from the coordinates of the old ones. The formalism was first developed by Anderson for a program that analyzes two-dimensional mathematical expressions [Anderson 1968] and was subsequently investigated by Milgram and Rosenfeld [1970]. The formalism is defined precisely here because it provides the basis for the symbolic characterization of shape grammars given in the next section.

The following definition is taken directly from [Milgram and Rosenfeld 1970]. A graphical rewriting grammar ("grammar with coordinates") is a 6-tuple <T, N, D, n, P, g> where

T is a finite set of terminal symbols

N is a finite set of non-terminal symbols, $T \cap N = \phi$

D is an infinite domain of "coordinates"

n is a positive integer, the number of coordinates used

P is a finite set of "productions", each of which is a 4-tuple <b, c, π, f>, where

- b is a j-tuple of symbols for some $j \geq 1$
- c is a k-tuple of symbols for some $k \geq 1$
- π is a predicate with k arguments, each of which is an n-tuple of coordinates
- f is a j-tuple of functions, each having k arguments; the arguments and function values are n-tuples of coordinates

g ∈ N is a special symbol, called the "goal" or initial symbol.

"A set S_{i+1} of symbols and associated n-tuples of coordinates is said to directly reduce into another such set S_i if there exists a production <b, c, π, f>, for which E is a subset of S_{i+1}; its coordinates satisfy π; the coordinates of the symbols of b are obtained from those in E by applying the functions in f; and $S_{i+1} - E \cup b = S_i$. Similarly, S" is said to reduce into S' if there exist $S'' = S_n, S_{n-1}, \ldots, S_1, S_0 = S'$ such that S_i directly reduces into S_{i-1}, $1 \leq i \leq n$. Finally, S is said to be a sentence of G if it reduces to {g} (with some associated coordinates). The set of all sentences whose symbols are all terminals is called the terminal language of G." [Milgram and Rosenfeld 1970]. Milgram and Rosenfeld point out that there is no gain in generality in taking n>1, i.e., any grammar with n>1 is equivalent to a grammar with n=1.

In his program to recognize two-dimensional mathematical expressions, Anderson used six coordinates for each terminal and non-terminal symbol: xmin, ymin, xcenter, ycenter, xmax, ymax, where (xcenter, ycenter) is the typographical center of the symbol. The graphical representation of one of Anderson's productions is shown in Figure 19a. This production is used to reduce a mathematical expression like the one in Figure 19b [Anderson 1968]. A similar formalism and program was developed by Chang [1970,1971].

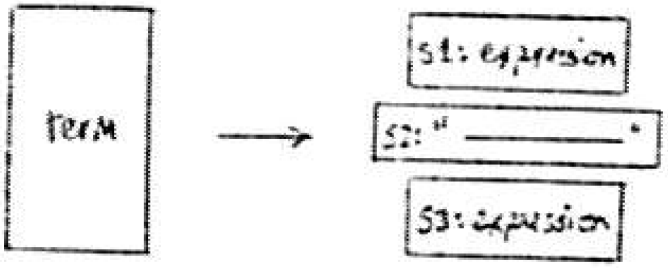


Figure 19a. Graphical representation of one of Anderson's productions (from [Anderson 1968]).

$$\frac{a^2+b}{c}$$

Figure 19b. Mathematical expression to which the production is applicable.

1.9 A symbolic characterization of shape grammars

The definition of shape grammars in Section 1.1 is designed to provide a straight-forward means of specifying a shape in terms of its underlying structure. The definition is visually oriented rather than symbolically oriented as this approach seems more natural considering the subject matter -- shapes. (For an interesting, if rather extremist, discussion of visual thinking vs. symbolic thinking see [Arnheim 1969]). Shape grammars are designed to be easily used and understood by people without the aid of a computer. It is however possible to characterize shape grammars symbolically; such a characterization is described here.

In this formalism, a shape is characterized as a set of symbols with associated parameters for specifying location, orientation, size, etc. Each symbol with associated parameters in such a set represents a different occurrence of a terminal or marker in the shape. Two disjoint sets of symbols are defined initially, one for the symbols representing terminals and one for the symbols representing markers. A shape rule is characterized as a predicate and a function. The predicate determines whether the shape rule is applicable to part of a shape. The function determines the effect of applying the shape rule if the shape rule is applicable.

The formalism for grammars with coordinates described in the last section provides the basis for the symbolic characterization of a shape grammar as a 6-tuple $<T, M, D, n, P, I>$ where

T is a finite set of terminal symbols

M is a finite set of non-terminal (or marker) symbols such that $T \cap M = \phi$

D is an infinite domain of parameters

n is a positive integer, the number of parameters to be associated with each symbol

P is a finite set of productions, each of which is a 4-tuple <b, c, π, f> where

b is a j-tuple of symbols for some $j \geq 1$

c is a k-tuple of symbols for some $k \geq 1$

π is a predicate with k arguments, each which is an n-tuple of parameters

f is a j-tuple of functions, each having k arguments; the arguments and function values are n-tuples of parameters

I is a set of pairs, each pair of the form <s,p> where $s \in T \cup M$ and p is an n-tuple of parameters.

Using this characterization, a shape S is a set of symbols and associated n-tuples of parameters, i.e., a shape is a set of pairs where each pair is of the form <s,p> with $s \in T \cup M$ and p an n-tuple of parameters. A production can be applied to a shape S only if S contains k symbols with associated parameters such that the symbols are identical to the symbols of c in the production and the associated n-tuples of parameters satisfy the predicate π of the production. A production is applied to a shape by replacing those k symbols and associated parameters by the j symbols of b with associated parameters as calculated by the j functions of f. A shape is derived by beginning with I and successively applying productions. A shape is in the language if it can be derived from I and contains no non-terminal symbols. Note that in this formalism productions are applied to the initial shape to obtain a

sentence in the language, whereas in Anderson's formalism productions are applied to a sentence in the language to reduce it to the goal symbol. Note also that the word "parameter" is used instead of "coordinate" as it seems more descriptive.

In the symbolic characterization of a two-dimensional shape grammar there would be one symbol in T for each shape in V_t and one symbol in M for each shape in V_m. The number n would be five. The five parameters associated with each symbol would include one for the x location of the symbol, one for the y location, one for the orientation, one for the scale, and one to indicate mirror image. For each shape rule there would be one production. k would be equal to the number of terminals and markers in the left side of the shape rule. c would contain the symbol for each of those terminals and markers. The predicate in the production would be true only if the subset of the shape under consideration were identical to the left side of the shape rule. j would be equal to the number of terminals and markers in the right side of the shape rule. b would contain the symbol for each of those terminals and markers. The j functions of f would calculate the proper parameters to associate with the j symbols in b to make the added shape the equivalent of the right side of the rule. I would contain one symbol with associated parameters for each terminal and marker in the initial shape of the shape grammar.

While this formalism may seem fairly complicated and it might be difficult to understand a shape grammar presented solely in this format, it does facilitate writing computer programs that make use of shape grammars. In particular, this characterization underlies the computer program described in Section 3.1.2.

SECTION 2 ANALYSIS OF SHAPES USING SHAPE GRAMMARS: A PROGRAM THAT USES A SHAPE GRAMMAR TO SOLVE A THREE-DIMENSIONAL PERCEPTUAL TASK

Shape grammars can be used in the analysis, as well as the generation, of shapes. Shape grammars provide a means for the explicit specification of the structure underlying shapes; it follows that shapes can be analyzed in terms of this structure. An important feature of the use of shape grammars as an analytic tool is that it allows for the analysis of complicated shapes that may never have been seen before, just as phrase structure grammmars can be used in the analysis of complicated and previously unknown strings of symbols (e.g. Algol programs).

In the past, picture grammars (of the various types) have been used exclusively in the analysis of pictures of an essentially two-dimensional nature (see Section 1.8). While shape grammars may well be useful in the analysis of pictures of this type, this problem is not investigated here. Instead, the more interesting question of the use of shape grammars in the analysis of pictures of three-dimensional objects is investigated. There has been some discussion on the utility of picture grammars in the analysis of pictures of three-dimensional objects, but to my knowledge, there previously have been neither actual computer programs that analyze pictures of three-dimensional objects in terms of explicit picture grammars nor concrete suggestions as to how to do this.

In this section, a computer program that uses shape grammars to solve a perceptual task involving the analysis of pictures of a very restricted class of three-dimensional objects is described. This example is not meant to be exhaustive; only one approach to the use of shape grammars in the automated analysis of shapes is explored.

2.1 The task

The perceptual task performed by the program was developed by Roger Shepard and Jacqueline Metzler of the Department of Psychology here at Stanford and is reported in [Shepard and Metzler 1971]. The task can be described as follows: given a pair of perspective line drawings such as those in Figure 20a - 20c, determine whether the drawings portray objects that are identical to each other in terms of three-dimensional shape (i.e., the drawings can be considered different views of the same object) or whether the drawings portray objects that are three-dimensional mirror-images of each other. The line drawings in Figure 20a and 20b portray different views of the same objects. The line drawings in Figure 20c portray objects that are three-dimensional mirror images of each other.

Shepard and Metzler administered this task to human subjects and recorded the amount of time required by the subjects to decide whether the portrayed objects were "the same" or "mirror image". Two kinds of "same" pairs were used: the drawings portrayed two views of the same object either rotated in the picture plane (as in Figure 20a) or rotated in depth (as in Figure 20b). Their (surprising) results are summarized as follows:

> The time required to recognize that two perspective drawings portray objects of the same three-dimensional shape is found to be (i) a linearly increasing function of the angular difference in the portrayed orientations of the two objects and (ii) no shorter for differences corresponding simply to a rigid rotation of one of the two-dimensional drawings in its own picture plane than for differences corresponding to a rotation of the three-dimensional objects in depth. [Shepard and Metzler 1971]

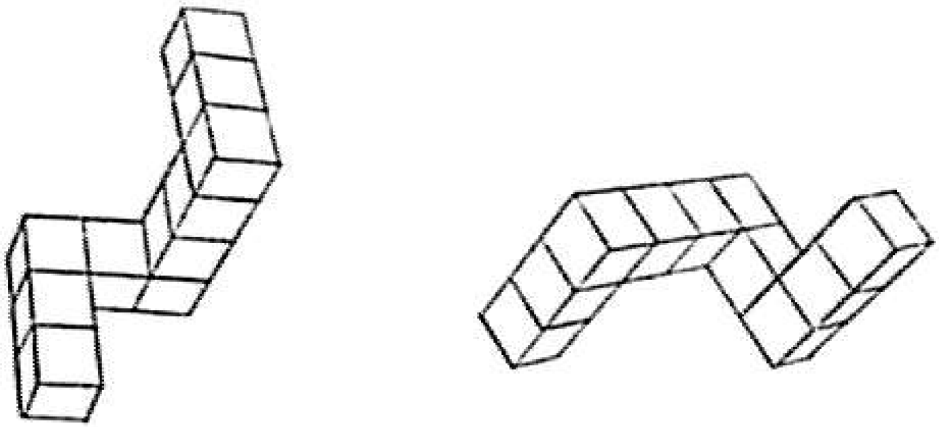


Figure 20a. Line drawings portraying identical objects rotated in picture plane.

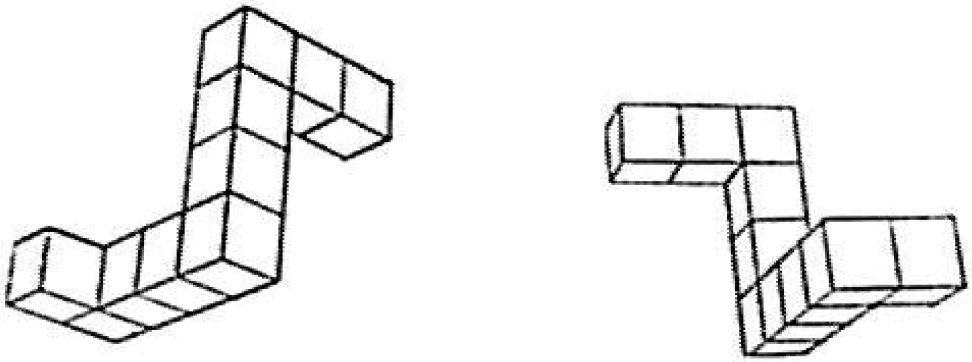


Figure 20b. Line drawings portraying identical objects rotated in depth.

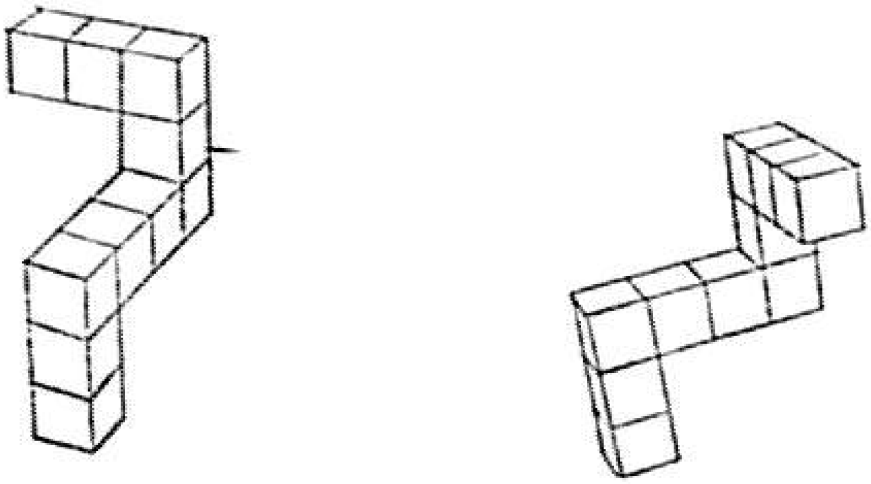


Figure 20c. Line drawings portraying mirror image objects.

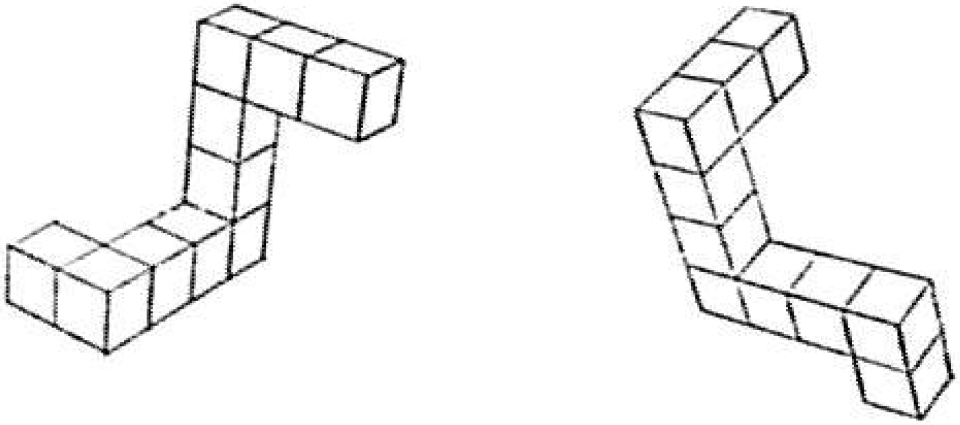


Figure 20d. Line drawings portraying structurally different objects.

These results seem to confirm the very subjective notion of solving the task by mentally rotating one of the portrayed three-dimensional objects at a fixed rate to determine if it matches the other portrayed object. The results suggest that it is just as easy to mentally rotate an object in depth as it is in the picture plane and that these rotations can be done at roughly 60 degrees per second [Shepard and Metzler 1971].

The task was expanded slightly for the computer program. The program is required to determine whether the pairs of drawings portray (1) identical objects (as in Figure 20a and 20b), (2) mirror images (as in Figure 20c) or (3) structurally different objects (as in Figure 20d). If the pairs are reported as identical or mirror image, the program also gives the three-dimensional axis equivalences. Shepard and Metzler always used pictures of objects consisting of strings of exactly ten cubes; there is no restriction on the number of cubes for the program. The program successfully analyzed the pair of line drawings shown in Figure 21 as well as those in Figure 20. It should be noted at the outset that there was no intention of simulating the processes used by people in solving this task.

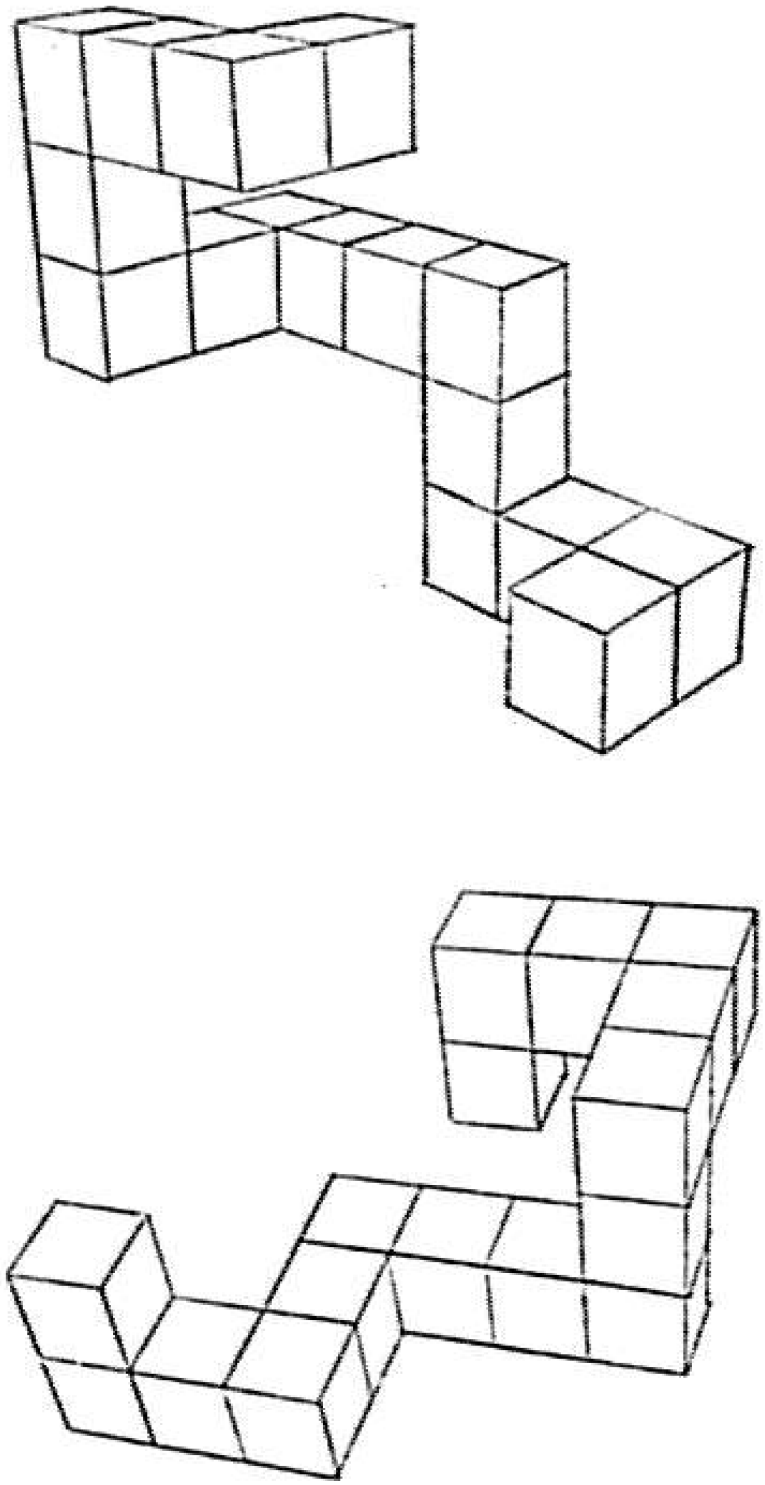


Figure 21. Another pair of line drawings successfully analyzed and compared by the program.

2.2 Program overview

The program that solves the perceptual task is written in SAIL [VanLehn 1973] and runs on the Stanford Artificial Intelligence Laboratory PDP-10. Input to the program are two files prepared using Geomed [Baumgart 1973], a geometric editor. Each file is a specification of a perspective line drawing of an object. Namely, each file is a list of the two-dimensional coordinates of the endpoints of each line segment occurring in the line drawing. There are four possible outputs for the program: (1) a statement that the objects portrayed by the line drawings are identical and an indication of the three-dimensional axis equivalences of the objects, (2) a statement that the objects portrayed by the line drawings are mirror images and an indication of the three-dimensional axis equivalences of the objects, (3) a statement that the objects portrayed by the line drawings are completely different, or (4) a statement indicating the failure of the program to successfully analyze one of the line drawings.

The program has three parts: (1) preprocessing, in which the vertices of the line drawing are classified, a data structure is constructed, and the three-dimensional axes are determined, (2) analysis and model building, in which the shape rules of the shape grammar are applied to the line drawing and a model of the object is constructed, and (3) comparison of models, in which the models constructed for the two objects are compared. The first two parts are applied to each line drawing independently, the idea being to construct a model of the three-dimensional structure of each portrayed object. In the third part the two models are compared to determine the relationship between the objects.

The process of analysis and model building in the program is the process of extracting the three-dimensional structure of the portrayed object from the two-dimensional line drawing. The model is constructed during application of the shape rules of the shape grammar to the line drawing. Each time a rule of the shape grammar is applied, something new is added to the model. This process can be likened to the syntactic analysis component of some compilers. The preprocessing is similar to the lexical scan. Applying a shape grammar to a line drawing is similar to parsing a symbol string using a phrase structure grammar. The model constructed for the portrayed object is not dissimilar to the tree representation that might be constructed by a compiler for an arithmetic expression. Just as the tree embodies the structure of the expression, the model embodies the structure of the object.

In the next three sections, a detailed description of the program is given. The particular shape grammar used and model constructed are presented in Section 2.4.

2.3 Preprocessing

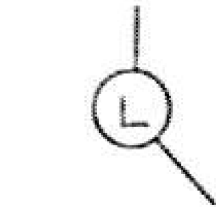
The preprocessing stage of the program includes building the data structure, classifying the vertices, and determining the three-dimensional axes.

The LEAP associative data structures [Feldman and Rovner 1969] of SAIL are used to represent the line drawing in the program. For each line segment of the line drawing, two associations of the form

$$\text{ENDPOINT} * L_i = V_j$$
$$\text{and} \quad \text{ENDPOINT} * L_i = V_k$$

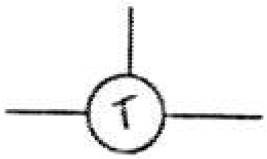
are added where L_i is an item representing the line segment and V_j and V_k are items representing the vertices of the line segment. Each vertex has a one-dimensional array of length two as a datum. The array contains the x and y coordinates of the vertex in the line drawing. As the associations are added, the coordinates of the vertices are compared with the coordinates of previously added vertices. Vertices that are located within a certain threshold distance are considered identical and the data structure is constructed accordingly.

After the associations for the line segments have been added, each vertex is classified in terms of the number of lines that meet at the vertex and the angles between the lines. Vertex classification roughly follows that of Guzman [1968]. Seven vertex types are allowed: L, W, T, Y, Psi, X, and 5. The definition of vertex types is given in Figure 22. Vertices of type T are further classified as either T1 or T3 using local information. If a T vertex is contained in a parallelogram of vertices (see Figure 23) it is classified as type T3 and is still considered a vertex

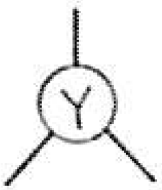


Type L: A vertex where two lines meet

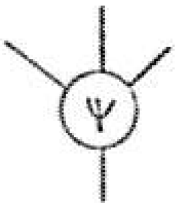
Type W: A vertex where three lines meet with one angle > 180 degrees



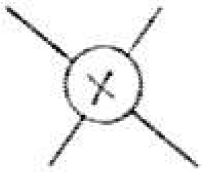
Type T: A vertex where three lines meet, one pair colinear (see Figure 23 for further classification)



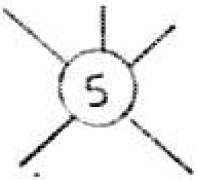
Type Y: A vertex where three lines meet with all angles < 180 degrees



Type Ψ: A vertex where four lines meet, one pair colinear



Type X: A vertex where four lines meet, two pairs colinear



Type 5: A vertex where five lines meet

Figure 22. Definition of vertex types.

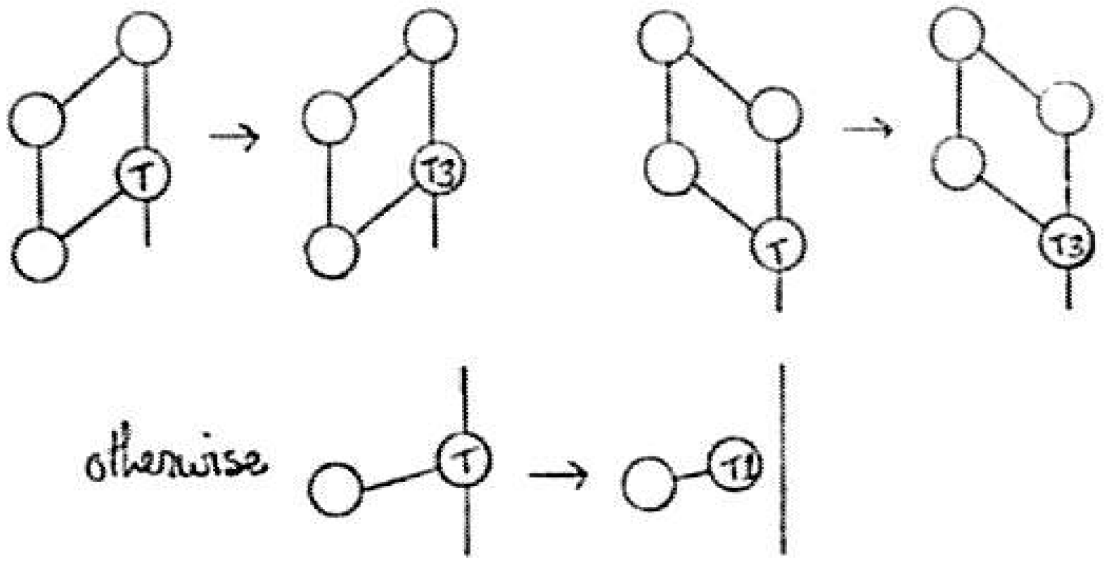


Figure 23. Classification of type T vertex into type T3 or type T1.

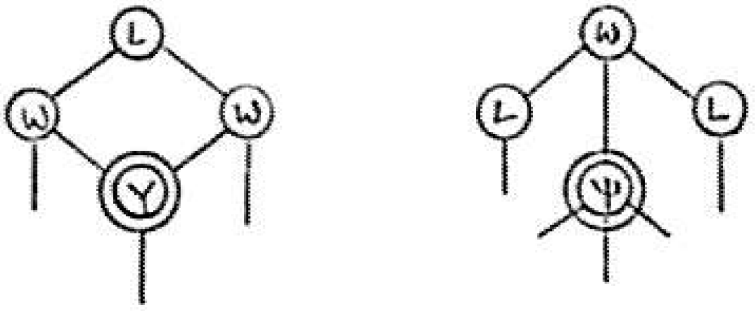


Figure 24. The two allowable vertex configurations for the initial double-circled vertex.

of three lines. If a T vertex is not contained in a parallelogram of vertices, it is classified as type T1, considered a vertex of one line (see Figure 23), and the ENDPOINT associations are changed accordingly. For each vertex, an association of the form

$$\text{VTYPE} \ll V_i \gg \text{vertex\_type}$$

is added, where V_i is the item for the vertex and vertex_type is the item for its vertex type.

One of the vertices of the line drawing is distinguished by surrounding it with an extra circle. Intuitively, this vertex is the central vertex of one of the end cubes of the object. The vertex is either of type Y or type Psi. The possible vertex configurations for this distinguished vertex are shown in Figure 24. If, as normally occurs, two different vertices of the line drawing qualify, i.e. one on each end cube, one of these vertices is chosen arbitrarily.

After the vertices are classified, a three-dimensional axis system for the line drawing is determined using the distinguished vertex. If this vertex is of type Y, the orientations of the three line segments radiating from the vertex are calculated. If the vertex is of type Psi, the orientations of the three line segments that would form a type Y vertex are calculated. These three orientations are assigned directions +x, +y, and +z in a counter-clockwise order. This insures that the axis system is a left handed system. The axis system will be used in specifying the model constructed for the portrayed object.

The effect of preprocessing on a line drawing portraying an object composed of six cubes is shown in Figure 25.

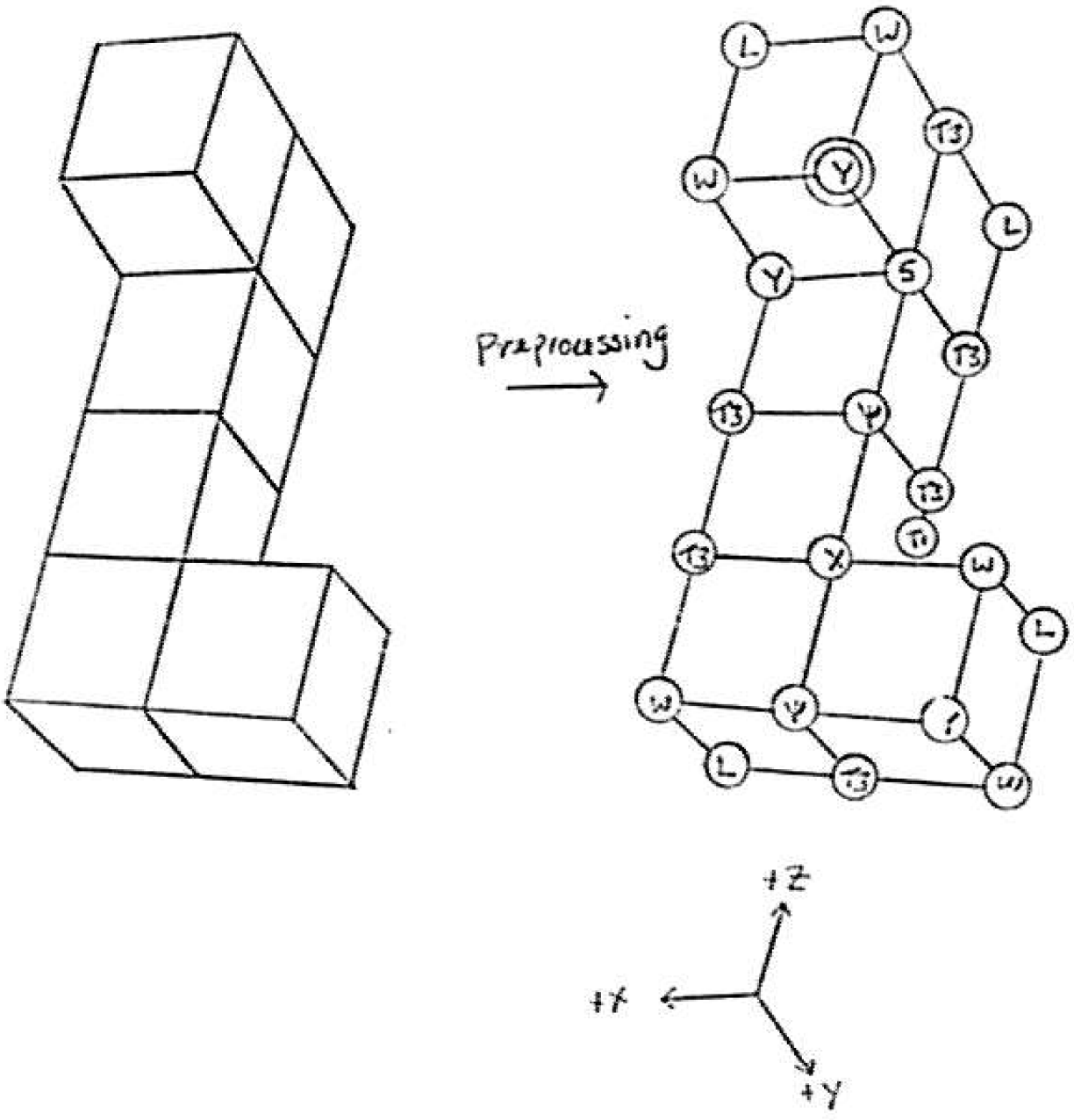


Figure 25. The effect of preprocessing on a simple line drawing.

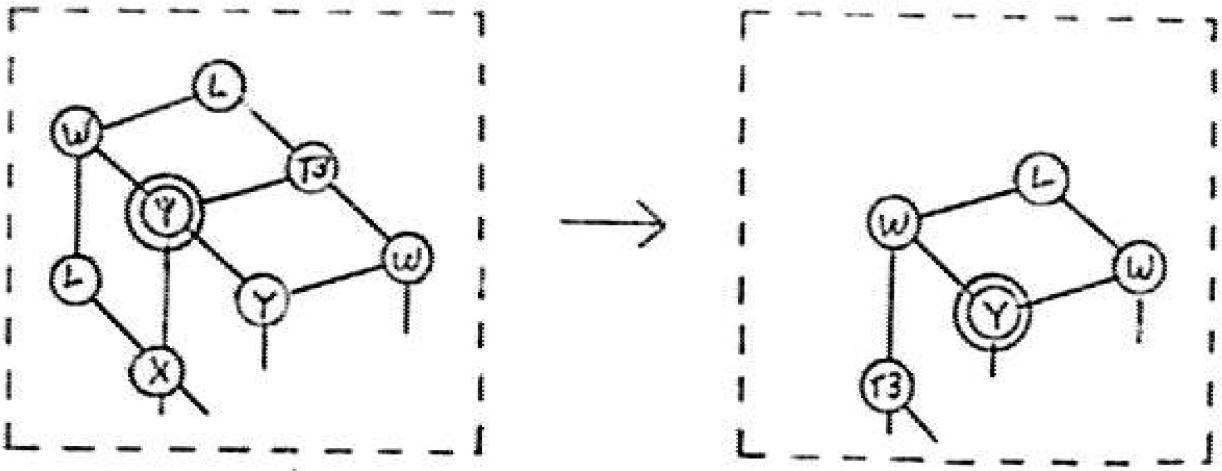
2.4 Analysis and model building

The heart of the program is the part that applies the shape grammar to the line drawing with classified vertices and constructs the model.

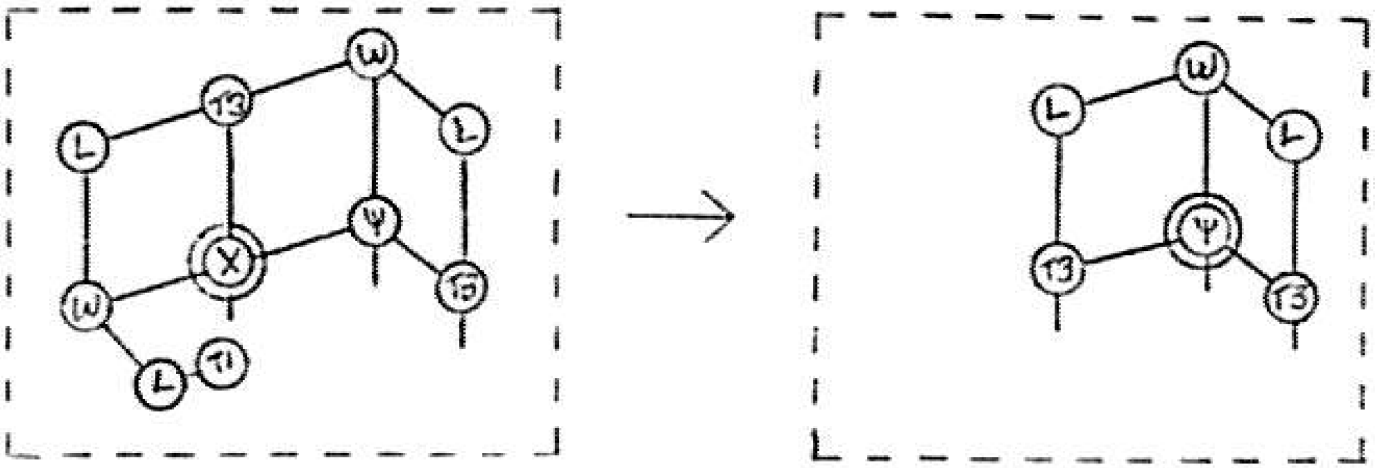
The shape rules of the shape grammar that is used to analyze the line drawings are shown in Figure 26. The nonterminals of the shape grammar are a circle and the symbols for the different vertex types. The terminal is a straight line. The shape grammar contains nine shape rules. The initial shape is the line drawing (with classified vertices) to be analyzed. Rather than beginning with a simple shape, adding terminals and markers during shape rule application, and generating the shapes of interest, this shape grammar begins with a shape of interest and erases terminals and markers during rule application until none remain. This approach effectively elides the parsing problem for shape grammars. The approach seems straightforward since the shape grammar will be used only in the analysis of shapes and never in the generation of shapes. A similar approach was taken by Anderson in the formalism described in Section 1.8.4.

A word should be said here about the parsing problem for shape grammars. The parsing problem can be defined as: given a shape and a shape grammar, determine whether the shape is in the language defined by the shape grammar and if it is, find a generation that yields the shape. The parsing problem is difficult for shape grammars for two reasons. First, for a given shape and a given set of terminals in a shape grammar, there may be many ways of dividing up the shape into terminals. Second, almost all interesting shape grammars (and all shape

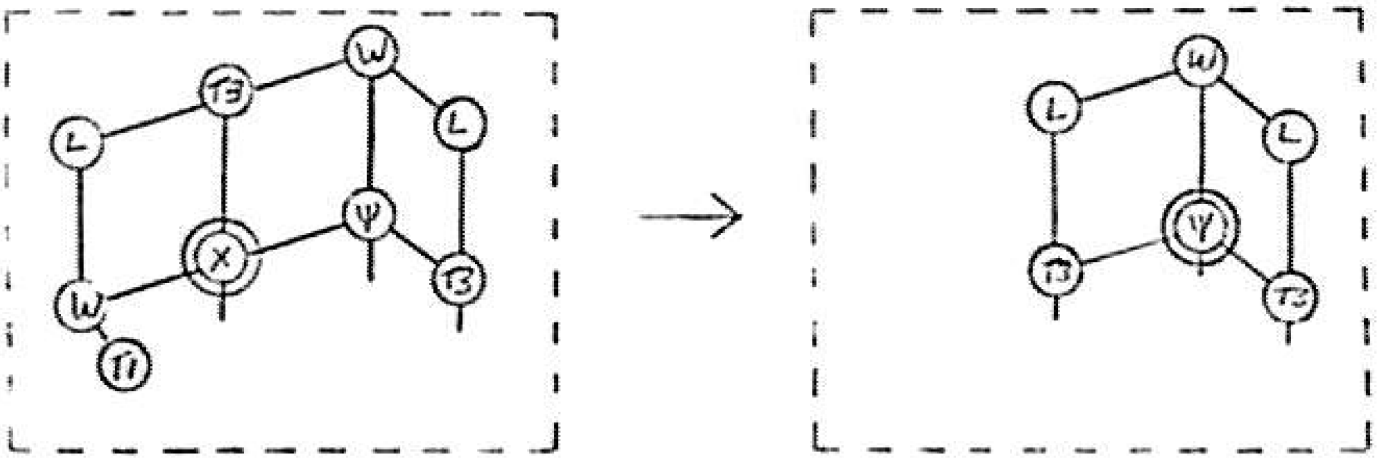
rule 1



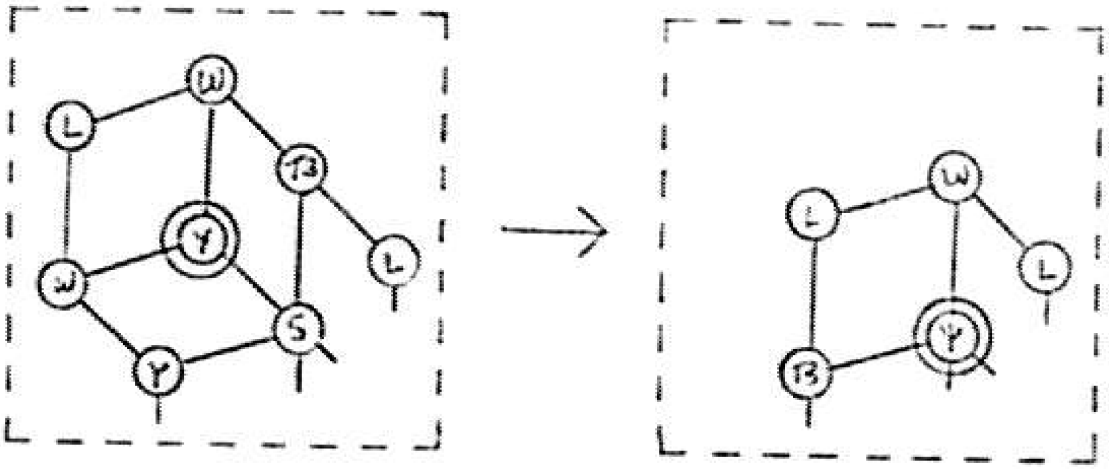
rule 2



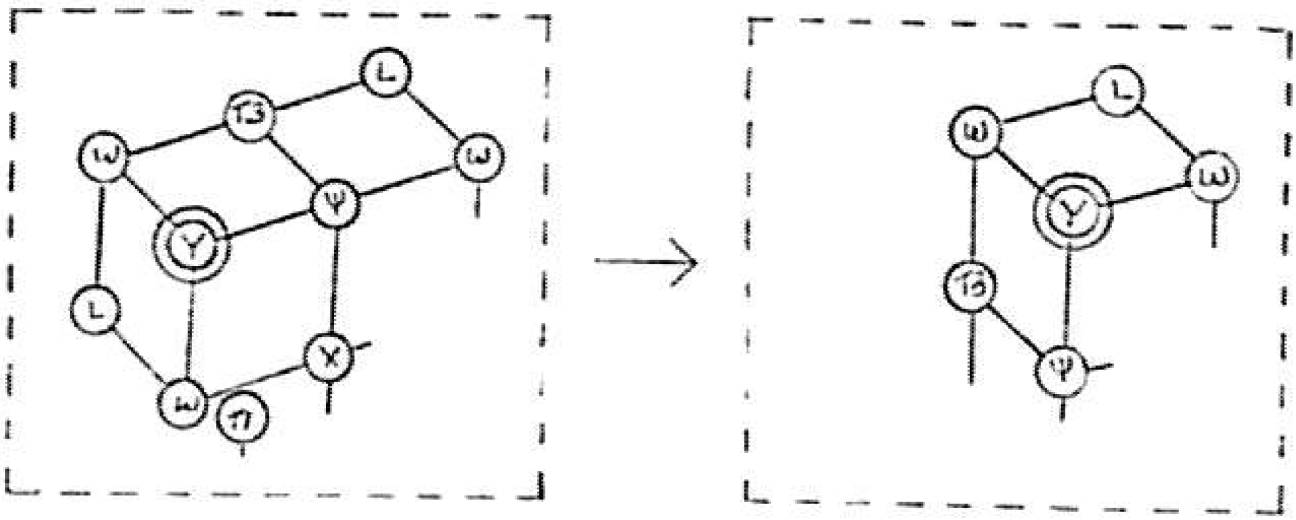
rule 3



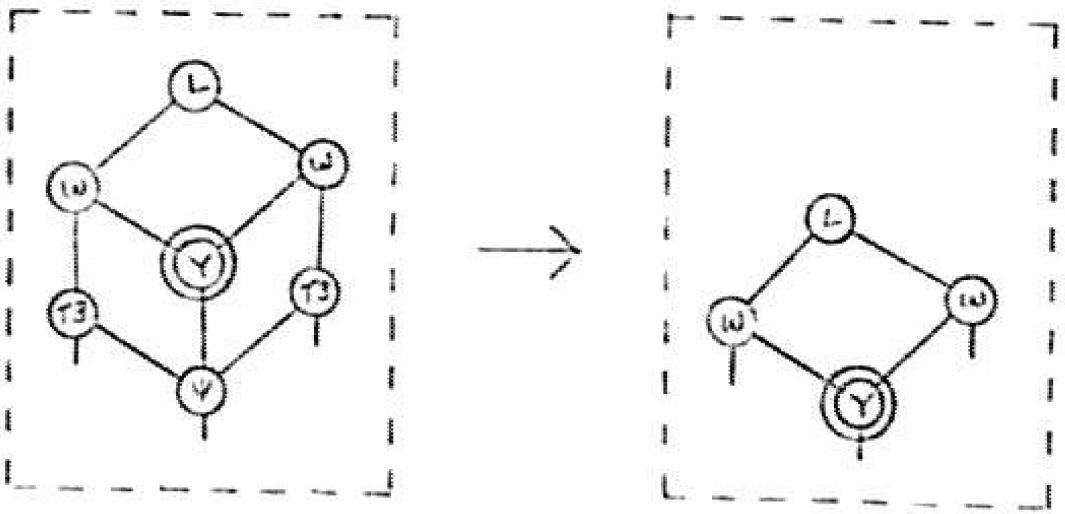
Rule 4



Rule 5



Rule 6



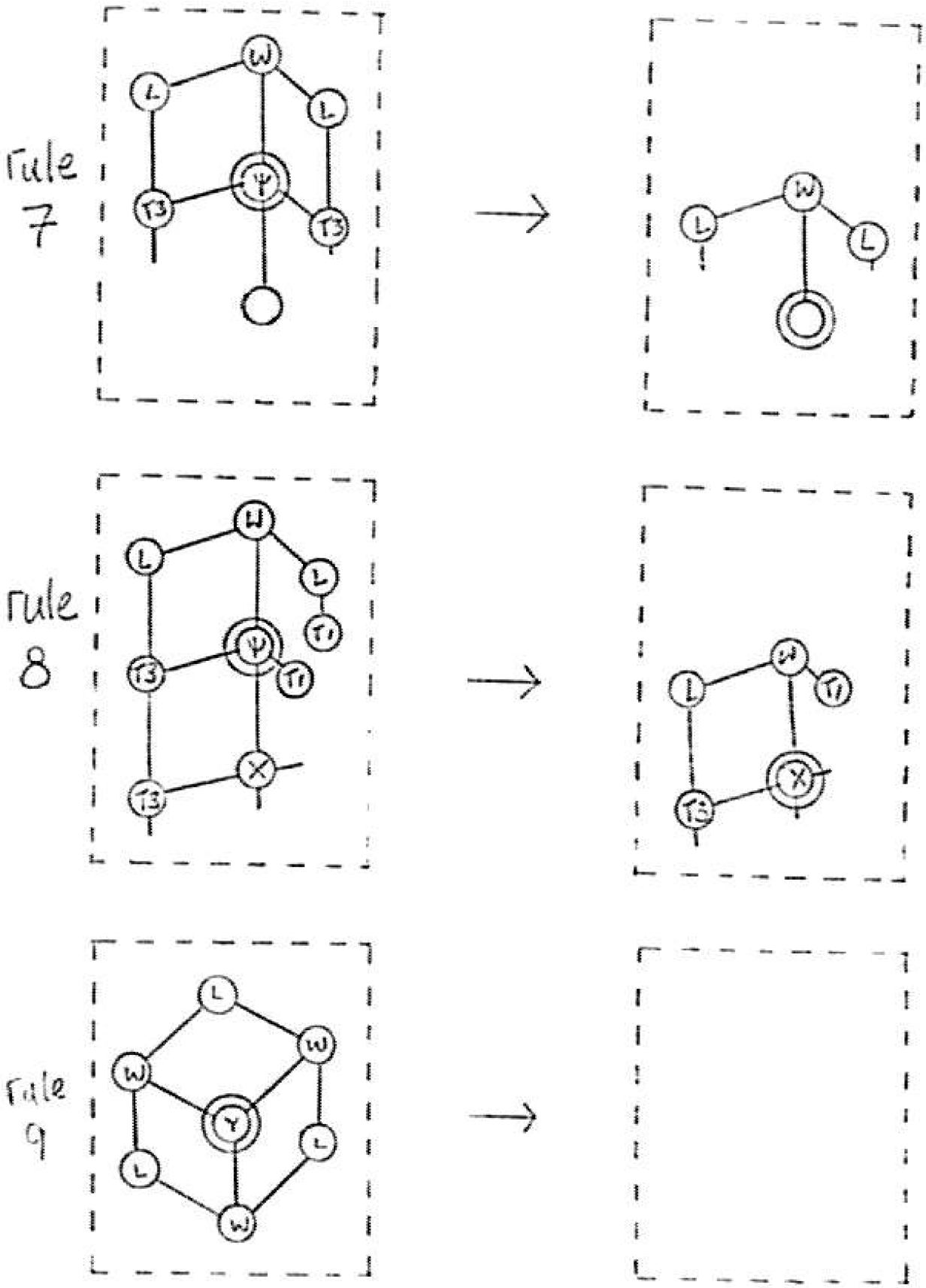


Figure 36. The shape rules used to analyze the line drawings.

grammars presented in this dissertation) are context-sensitive in the sense that terminals are present in the left sides of shape rules. There are no known efficient and general parsing algorithms for context-sensitive phrase structure grammars; there is no reason to believe the problem is any easier for shape grammars. It should be noted that for non-erasing shape grammars (see Section 1.2), the parsing problem is solvable by enumeration, in theory.

Returning to the shape grammar for the Shepard-Metzler figures, notice that the left sides of the shape rules contain different vertex configurations that can portray an end cube. These different vertex configurations result from the various possible three-dimensional structures of the objects and the various viewpoints from which these objects can be seen. The right sides of the shape rules are the vertex configurations that result from the removal of the end cube. As usual, the "generation" process begins with the initial shape, the line drawing with classified vertices, and consists of the repeated application of the shape rules. During the process of shape rule application, the line drawing is erased one cube at a time. The process terminates when no rules are applicable. In the preprocessing stage, one of the vertices of one of the end cubes was marked with an extra circle. The left side of each of the shape rules contains a double-circled vertex. Thus each shape rule is only applicable to the portion of the line drawing containing the double-circled vertex. Each application of a shape rule results in the elimination of the vertex configuration which portrays the end cube with a double-circled vertex, the placement of an extra circle around a vertex of the next cube, and the filling in of edges and vertices which were obscured by the erased cube.

The shape rules shown in Figure 26 are actually shape rule schemata. In the application of the rules, the exact angles between the lines at each vertex are ignored within the bounds of the definition of the vertex type. All that matters is the vertex configurations. Also, strictly speaking the marker indicating vertex type within each circle should be in a standard orientation relative to the lines that meet at the vertex. For convenience, these vertex markers are drawn vertically in the figures. As with all shape grammars, the transformations of rotation, scale, translation, and mirror image may be applied to the left and right sides of the rules during rule application.

The rules of the shape grammar are embodied in SAIL code in the program. First the code establishes which rule is applicable by determining which vertex configuration of the left side of a rule exists in the current shape as specified by the current associations. The rule is applied by changing the associations to match the right side of the rule. This may result in the addition, as well as the deletion, of line segment and vertex items and associations. This section of code is repeated until no rules are applicable. If at this point no line segment or vertex items and associations remain, the program has succeeded in analyzing the line drawing. If no rules are applicable but some items and associations remain, the program has failed.

If this part of the program consisted solely of an implementation of the shape grammar, the program could be used only as an acceptor. Namely, the program could make a binary choice: it could determine whether or not a line drawing does indeed portray a Shepard-Metzler object (as defined by the shape grammar) by

classifying the vertices and checking if the shape grammar could be applied until no terminals or markers remained. But the task involves extracting the three-dimensional structure of the objects portrayed by the line drawings. To do this, a model of the portrayed object is constructed during shape rule application.

The model used consists of a one-dimensional array of characters which specifies the skeleton of the portrayed object. The skeleton can be thought of as the sequence of line segments connecting the centers of the cubes. The model consists of the sequence of directions of the skeleton, where the directions are the axis directions determined during preprocessing. There are six possible directions: +x, -x, +y, -y, +z, -z. The model is constructed by recording the direction of the extra circle. Each time a rule is applied (except for the final application of rule 9) the axis direction closest to the direction of motion of the extra circle is added to the model. The length of the model is one less than the number of cubes in the portrayed object.

An example of this process is shown in Figure 27 for the line drawing of Figure 25. The model constructed for the object portrayed by this line drawing is +y -z -z -z -x. The models constructed for the objects portrayed by the line drawings shown in Figure 20 and 21 are shown in Figure 28. In the construction of the models in this figure, the convention that the uppermost of two eligible vertices is distinguished as the initial double-circled vertex is used in preprocessing.

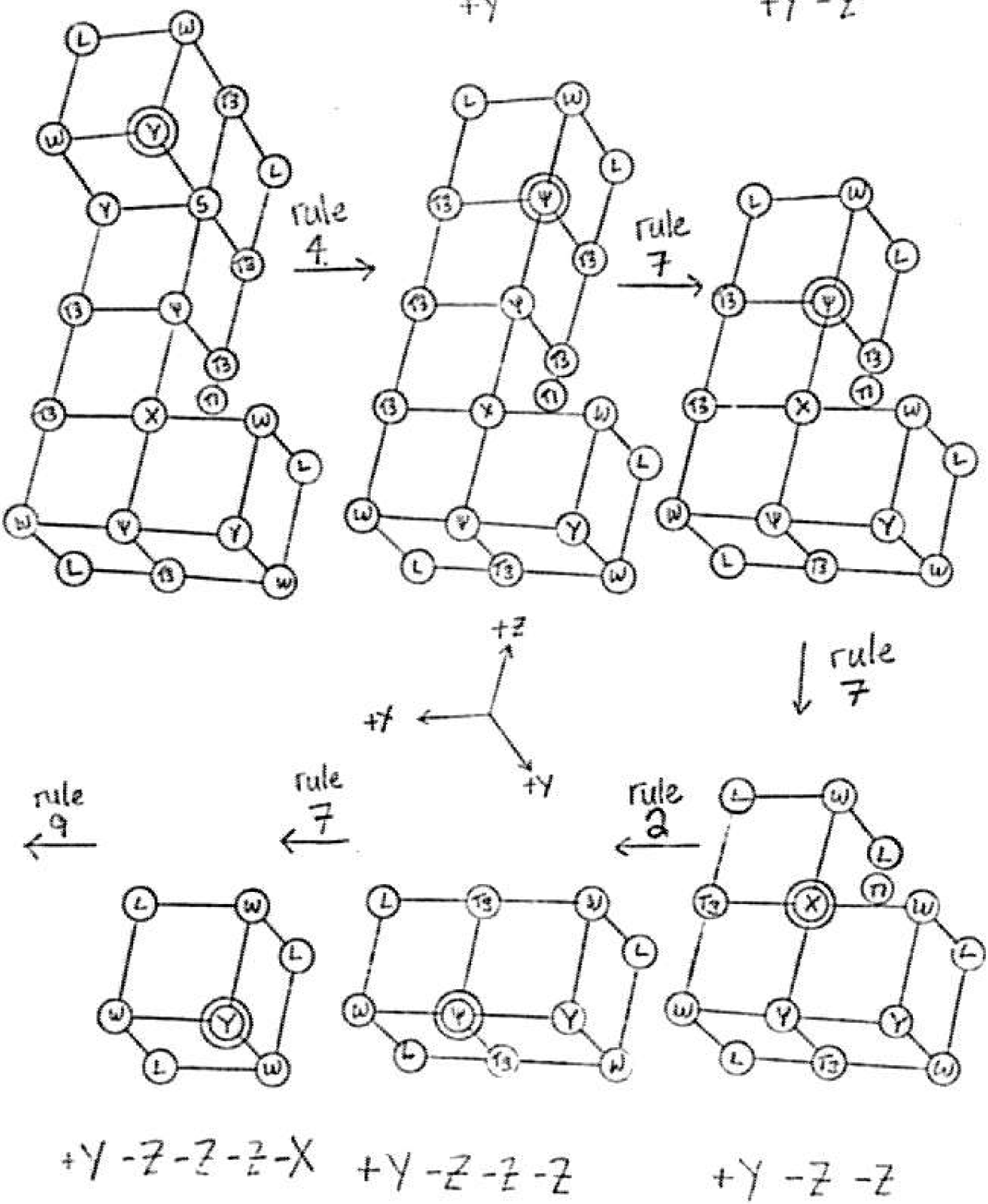


Figure 27. Rule application and model construction for the line drawing of Figure 25.

20a
left

-Y -Y +Z +Z +Z -X -X -Y -Y

20a
right

+Z +Z +Y +Y -X -X -X +Z +Z

20b
left

-Z -Z -X -X -X +Y +Y +Y -Z

20b
right

+X +X -Y -Y -Y -Z -Z -Z +X

20c
left

-Y -Y +Z +Z -X -X -X +Z +Z

20c
right

+Y +Y +Z +Z -X -X -X +Z +Z

20d
left

+Y +Y +Z +Z +Z -X -X -X +Y

20d
right

-X -X +Z +Z +Z -Y -Y -Y +Z

21
left

-X +Y +Y +Z +Z +X +X -Y -Y -Y +Z +Z -Y -Y -X

21
right

+Z -Y -Y +X +X -Y -Y -Y -Z -Z +X +X +Y +Y +Z

Figure 28. The three-dimensional axes and models constructed for the line drawings of Figures 20 and 21.

2.5 Comparison of models

After the preprocessing and model building routines are applied to each line drawing independently, the models that were constructed for the portrayed objects are compared. The model comparison has two stages.

The first stage determines whether the two models represent similar (i.e. identical or mirror image) objects or completely different objects. Recall that each model is a one-dimensional array of axis directions where the possible directions are +x, -x, +y, -y, +z, and -z. First, the lengths of the two models are compared. If the lengths are unequal, the objects are declared to be different and the model comparison routine is terminated. If the lengths are equal, the program attempts to find axis equivalences for the +x, +y, and +z directions of the first model.

An axis direction d_1 of the first model is considered equivalent to an axis direction d_2 of the second model (written $d_1 \equiv d_2$) iff

> (1) the positions in which d_1 occurs in the first model are identical to the positions in which d_2 occurs in the second model and
>
> (2) the positions in which $-d_1$ occurs in the first model are identical to the positions in which $-d_2$ occurs in the second model.

For example, in the models shown in Figure 28 derived for the line drawings of Figure 20b, the axis equivalences for the +x, +y, and +z directions of the first model are $+x \equiv +y$, $+y \equiv -z$, and $+z \equiv -x$. Note that by the definition of axis equivalence, if $d_1 \equiv d_2$ then $-d_1 \equiv -d_2$.

If an equivalence can be found for each of the axis directions then the

portrayed objects must be similar (identical or mirror image) and the second stage of the model comparison routine is entered. If an equivalence for each of the directions cannot be found, no final conclusions can be drawn. The objects still could be similar if the initial double-circled vertices were located in preprocessing at different ends of the objects. This occurred, for example, with the construction of the models in Figure 28 for the line drawings in Figure 20a. To determine if the portrayed objects are similar or different, the second model is reversed and axis equivalences for the +x, +y, and +z directions of the first model again are searched for. To reverse the second model, each entry in the model is first negated and the sequence of directions is reversed. For example, the reverse of the model +z +z +y +y -x -x -x +z +z for the right line drawing of Figure 20a (see Figure 28) is -z -z +x +x +x -y -y -z -z. This new model is the model that would have been constructed if the initial double-circled vertex had been located at the other (lower) end of the object and the model had been constructed in reverse order. If, again, an axis equivalence cannot be found for each of the directions of the first model, the objects are declared different and the model comparison routine is terminated. If equivalences are found, the objects must be either identical or mirror image and the second stage is entered.

The second stage of the model comparison determines whether models with axis equivalences portray identical or mirror image objects. In the program +1 represents the +x direction, -1 represents -x, +2 represents +y, -2 represents -y, +3 represents +z, and -3 represents -z. Integer array EQUIV[1:3] is defined. The value of EQUIV[1] is the axis equivalence of the +x direction of the first model,

etc. By the definition of axis equivalence, the array EQUIV can have 6*4*2 = 48 different permutations of assigned values. For the models of Figure 20b, EQUIV[1] = +2, EQUIV[2] = -3, and EQUIV[3] = -1. Half of the 48 permutations represent axis equivalences that would occur if the second line drawing portrayed simply a three-dimensional rotation of the object portrayed by the first line drawing. The other half of the 48 permutations occur if the object portrayed by the second line drawing is a mirror image of the object portrayed by the first. The following SAIL subroutine determines whether the line drawings portray objects which are identical or mirror image:

```
INTEGER PROCEDURE SAME_OR_MI (INTEGER ARRAY EQUIV);
    BEGIN
    INTEGER I, MINUSAXES, DIFFAXES;
    MINUSAXES ← DIFFAXES ← 0;
    FOR I ← 1, 2, 3 DO
        BEGIN
        IF ABS(EQUIV[I]) ≠ I THEN DIFFAXES ← DIFFAXES+1;
        IF EQUIV[I] < 0 THEN MINUSAXES ← MINUSAXES+1;
        END;
    IF (MINUSAXES = 1) ∨ (MINUSAXES = 3)
    THEN RETURN (IF DIFFAXES = 2 THEN SAME ELSE MI)
    ELSE RETURN (IF DIFFAXES = 2 THEN MI ELSE SAME);
    END;
```

The subroutine determines whether the axis equivalences can be effected with just a three-dimensional rotation of the first object or whether an axis inversion (mirror image) is required. Recall that the method for determining the axis directions in preprocessing insures that both axis systems are left handed. The subroutine determines whether after transforming the first model into the

second model the resulting axis system would be left handed or right handed. If the axis system would still be left handed, the line drawings portray identical objects from possibly different viewpoints. If the axis system would become right handed, the line drawings portray objects that are mirror images of each other.

2.6 Program results

The range of line drawings that the program can analyze is defined effectively by the shape rules of Figure 26. The program has successfully compared over a hundred pairs of line drawings of objects.

It may be of interest to compare the performance of the program with the performance of people on the same task.

The program cannot analyze many line drawings that are analyzable by people. In particular, there are three types of views of an object for which the program fails: (1) a view of the object in which a substantative part of the object is obscured (as in Figure 29a), (2) a view of the object in which two different lines meet and appear to be one or two different vertices appear to coincide (as in Figure 29b), and (3) a degenerate view of the object in which only one face of a particular cube is observable (as in Figure 29c). Some of these problems could be fixed by not too complicated program patches. In particular, most degenerate views (of type 3) probably could be analyzed successfully with the addition of some rules to the shape grammar.

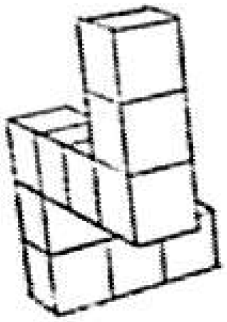


Figure 29a.
A view in which a substantive part is obscured.

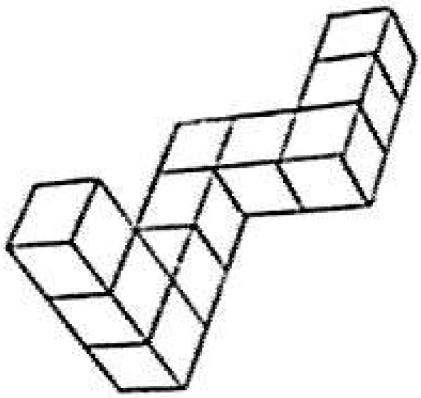


Figure 29b.
A view in which two different vertices appear to coincide.

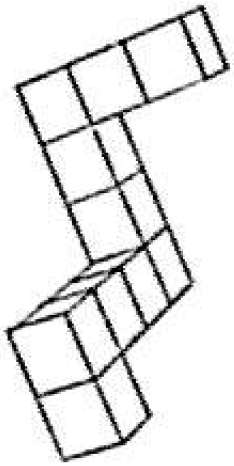


Figure 29c.
A degenerate view.

Figure 29. Three line drawings which the program cannot analyze.

The program succeeds in analyzing some line drawings which are very difficult for people, namely line drawings of objects containing many cubes, such as the line drawings in Figure 21. There is essentially no limit to the number of cubes in a line drawing that can be analyzed by the program. As Shepard and Metzler only used line drawings of objects containing ten cubes, no information was collected as to the limit of the number of cubes in line drawings analyzable by people. Personally, the solution of the task for the pair of line drawings in Figure 21 seems very much more difficult than the solution for the pairs in Figure 20. Mentally solving the task for line drawings portraying twice as many cubes as those portrayed in Figure 21 seems out of the question for me using the normal, non-analytic, mental rotation technique.

Shepard and Metzler were interested in the amount of time needed to solve the task. As noted in section 2.1, the amount of time required by people varies linearly with the angular difference of the portrayed orientations of the two objects. The amount of time taken by the program is independent of the angular difference of the portrayed orientations and is relatively constant for line drawings of objects of a fixed number of cubes. Indeed, where people would immediately recognize that two identical line drawings portray identical objects, the program would take about the same amount of time to solve this particular task as any other.

The input format for the program is different than that used for people. People are shown the line drawings. The program is given a digitized encoding of the line drawing made in terms of the coordinates of the endpoints of the line

segments. A front end program could have been written that made use of a television camera and a line finder algorithm. It is interesting to note that the program would take substantially longer to solve the task if shown the actual line drawing rather than given the encoded version while a person would certainly take longer to solve the task given the encoding rather than shown the line drawing itself. Shown the line drawing, the program would first encode it in terms of the coordinates of the vertices. Given the encoding, a person would first draw the picture.

All of this discussion must be tempered by the obvious fact that the program can solve only this one perceptual task while a person is a very general perceptual system.

2.7 Limitations and possible extensions

The task domain of the program was carefully chosen. The task has some important restrictions that greatly facilitate its solution. For example: The program is given encodings of perfect line drawings. Each line drawing portrays exactly one object. The range of objects the line drawings can portray is limited to objects composed of linear strings of cubes. The task can be solved using only structural information extractable from the line drawings.

A first extension to the program might be to add shape rules that would allow a cube to be attached to more than two other cubes. The objects to be analyzed

would no longer be restricted to strings of cubes but could be arbitrary assemblies of attached cubes. A one-dimensional array of directions would no longer suffice as a model of the objects. The new model constructed using this shape grammar might be a connected graph where the nodes would represent the centers of the cubes and the arcs of the graph would indicate adjacency or attachment of the cubes and be labelled as to the axis direction between the cubes. This model would still be a representation of the skeleton of the object. The model used in the program could be considered a special case of this connected graph model. More generally, the objects could be composed of not just cubes but a variety of primitive objects, e.g. wedges, prisms, octahedrons. The models constructed would be connected graphs as above but with each node labelled to indicate the type of object represented. Using this scheme, the process of determining whether two pictures portray different views of identical objects would consist of analyzing the pictures using the shape grammars, constructing the graph models, and determining whether the two graphs represent skeletons of identical objects. To help insure that shape rules are not applicable in unintended situations, a labelling technique such as the one described by Waltz [1972] might be used to label the portrayed edges of the object as concave, convex, etc. The lines between vertices in the shape rules would be labelled accordingly. Perhaps, lines between vertices could be allowed to be curves as well as straight lines. How large a class of objects can be analyzed using this type of technique is an open question.

The major advantage of this technique is that line drawings portraying a large number of objects can be analyzed, including line drawings portraying objects which

may never have been previously seen, using a fixed set of rules. Instead of requiring an object to be analyzed to be a member of a small, fixed class of objects, it is only required that the object be decomposable into instances of a fixed set of primitive objects attached in certain ways.

SECTION 3 GENERATION OF SHAPES USING SHAPE GRAMMARS: COMPUTER ART AND AESTHETICS

Shape grammars were developed originally as the basis for a formalism ("generative specifications") for generating non-representational, geometric paintings [Stiny and Gips 1972]. During the past four years, dozens of paintings have been specified using generative specifications and constructed using traditional artistic techniques. The process of using a generative specification to generate a painting has been implemented on the computer. Hundreds of paintings have been defined and displayed using this program.

This application of shape grammars is of interest (to me, at least) for two reasons:

(1) I like the paintings. I enjoy creating the paintings, looking at them hanging on my walls, and looking at them on the computer display. The paintings are of interest in themselves.

(2) The paintings provide a good initial problem domain for studying aesthetics and design.

The use of shape grammars in generating paintings has led to a study of aesthetics [Gips and Stiny 1973, submitted for publication]. An attempt has been made to define a formalism ("aesthetic systems") that can be used in specifying aesthetic viewpoints. A particular aesthetic system has been developed for paintings definable using generative specifications and has been implemented on the computer. The computer can evaluate paintings it displays in terms of this aesthetic viewpoint. Plans for an automatic design program are discussed. This work is intended as a step on the long path toward Artificial Intelligence programs of the type Firschein, et. al. [1973] call a "Creation and Evaluation System", namely

programs "capable of creative work in such areas as music, art (painting, scupture, architecture), literature (essays, novels, poetry), and mathematics and able to evaluate the work of humans" (and presumably other programs).

This section is divided into two parts. In Section 3.1, a formalism which uses shape grammars to generate paintings is defined, its computer implementation is described, and several examples are given. In Section 3.2, results of an investigation of aesthetics are presented and some of its ramifications are discussed.

3.1 Specification of painting using shape grammars

3.1.1 Generative specifications

A generative specification is a complete specification of a class of non-representational, geometric paintings. The primary component of a generative specification is a shape grammar. The paintings defined by generative specifications can be considered material representations of shapes generated by two-dimensional shape grammars.

A generative specification has four parts: (1) a shape gramar which defines a language of two-dimensional shapes, (2) a selection rule which selects shapes in that language for painting, (3) a list of painting rules which determine how the areas contained in the shapes are to be painted, and (4) a limiting shape which

determines the size and shape of the canvas and where the shapes are to be painted on the canvas.

In Sections 3.1.1.1 - 3.1.1.4 the four parts of a generative specification are defined and explained. The painting Triad, shown in Figure 30a, and its generative specification, shown in Figure 31a, are used as an example.

3.1.1.1 Shape grammar

Shape grammars used in generative specifications are generally non-erasing (see Section 1.2) and may be either serial or parallel. Recall that a serial shape grammar is denoted SGn and a parallel shape grammar PSGn (see Section 1.3). Serial shape grammars used in generative specifications are usually unimarker.

The shape grammar in the generative specification of Figure 31a has one terminal, a 60 degree arc of a circle, and one marker, two equal line segments joined at a 60 degree angle. There are four shape rules. The initial shape consists of six terminals which form a circle and one marker. A generation of a shape in the language is shown in Figure 31b. The shape grammar is somewhat similar to the (serial) shape grammar for the Snowflake curve shown in Figure 10 and discussed in Section 1.6.1. During the generation, rules 1 and 2 cause the marker to trace clockwise around the most recently added closed curve of terminals. Rule 1 is applicable when the marker is on the convex side of the next terminal arc. Application of rule 1 causes four terminals (i.e., a 240 degree arc) to

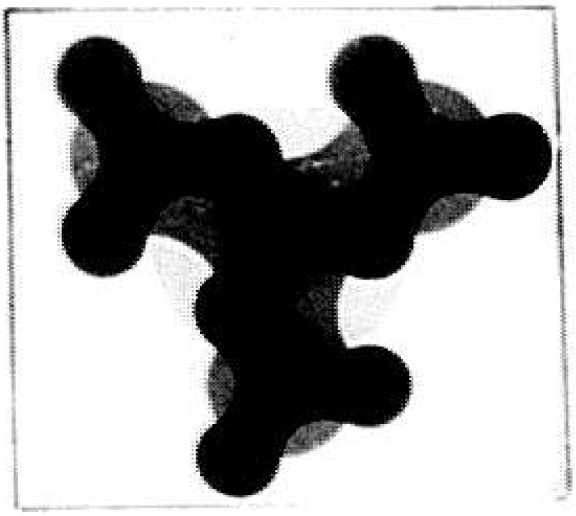


Figure 30a. Triad.
Colors are blue, red, yellow, and white (darkest to lightest).

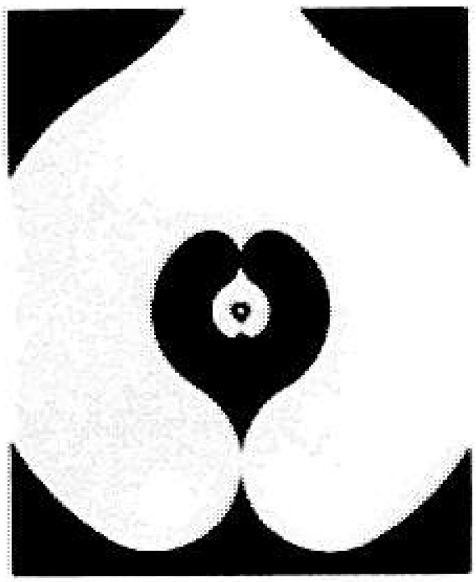


Figure 30b. Eve.
Colors are blue and white.

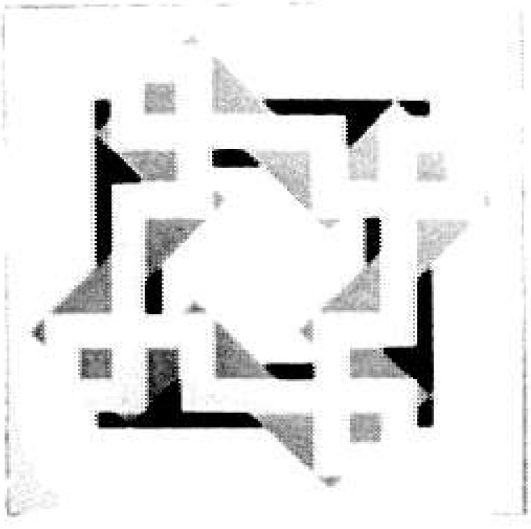


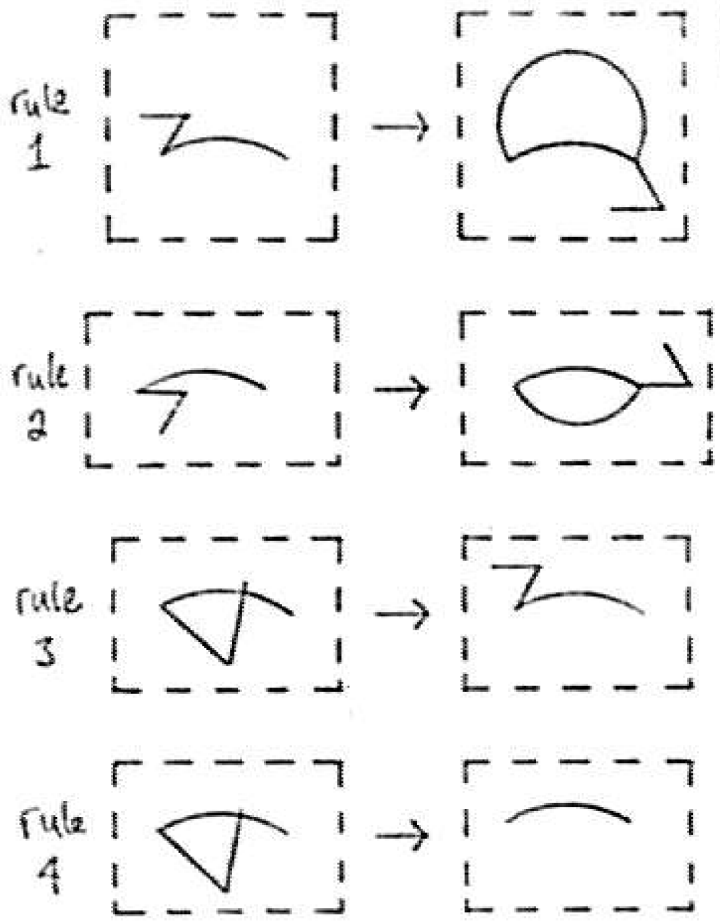
Figure 30c. Yellow Cross.
Colors are blue, red, yellow, and white (darkest to lightest).

Figure 30. Three paintings defined using generative specifications.

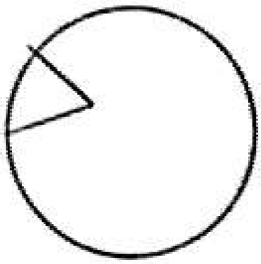
SHAPE GRAMMAR SG12

$V_T = \{ \frown \}$ $V_{NT} = \{ \wedge \}$

R contains



I is



SELECTION RULE : 2

PAINTING RULES

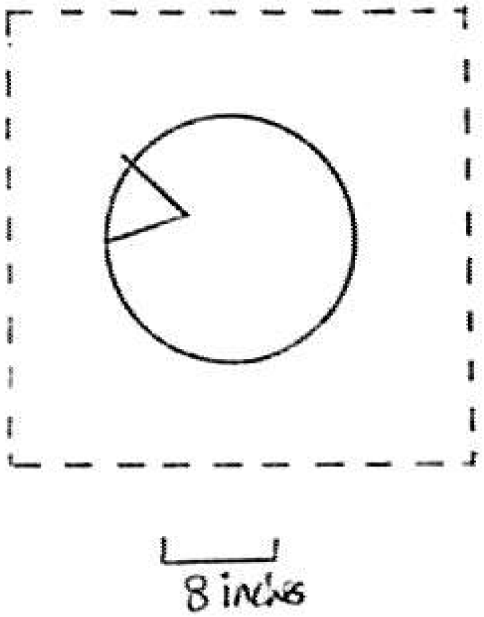
$L_2 \Rightarrow$ BLUE

$L_1 \cap \sim L_2 \Rightarrow$ RED

$L_0 \cap \sim (L_1 \cup L_2) \Rightarrow$ YELLOW

$\sim (L_0 \cup L_1 \cup L_2) \Rightarrow$ WHITE

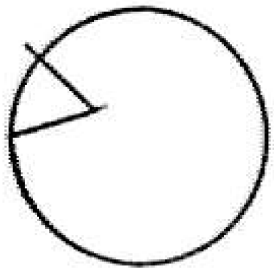
LIMITING SHAPE



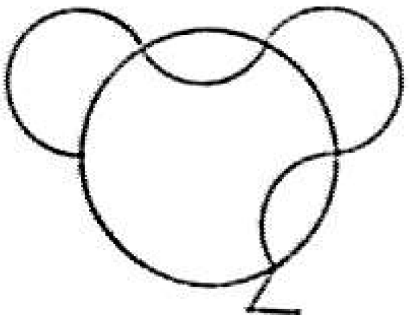
8 inches

Figure 31a. Generative specification for Triad.

INITIAL SHAPE

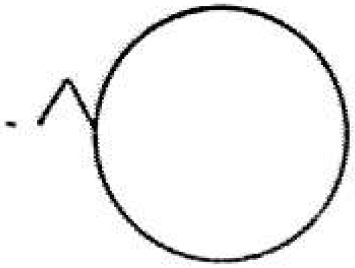


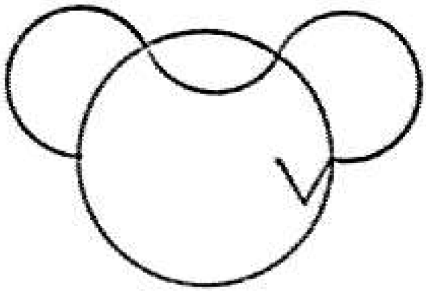
rule 1 →



↓ rule 3

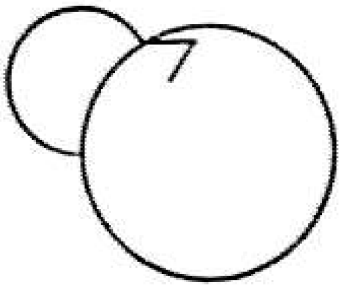
rule 2 ↑



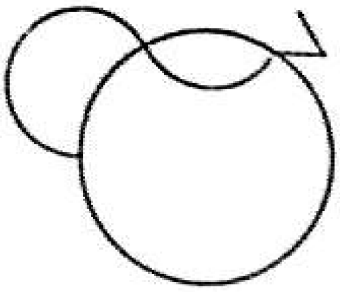


↓ rule 1

rule 1 ↑



rule 2 →



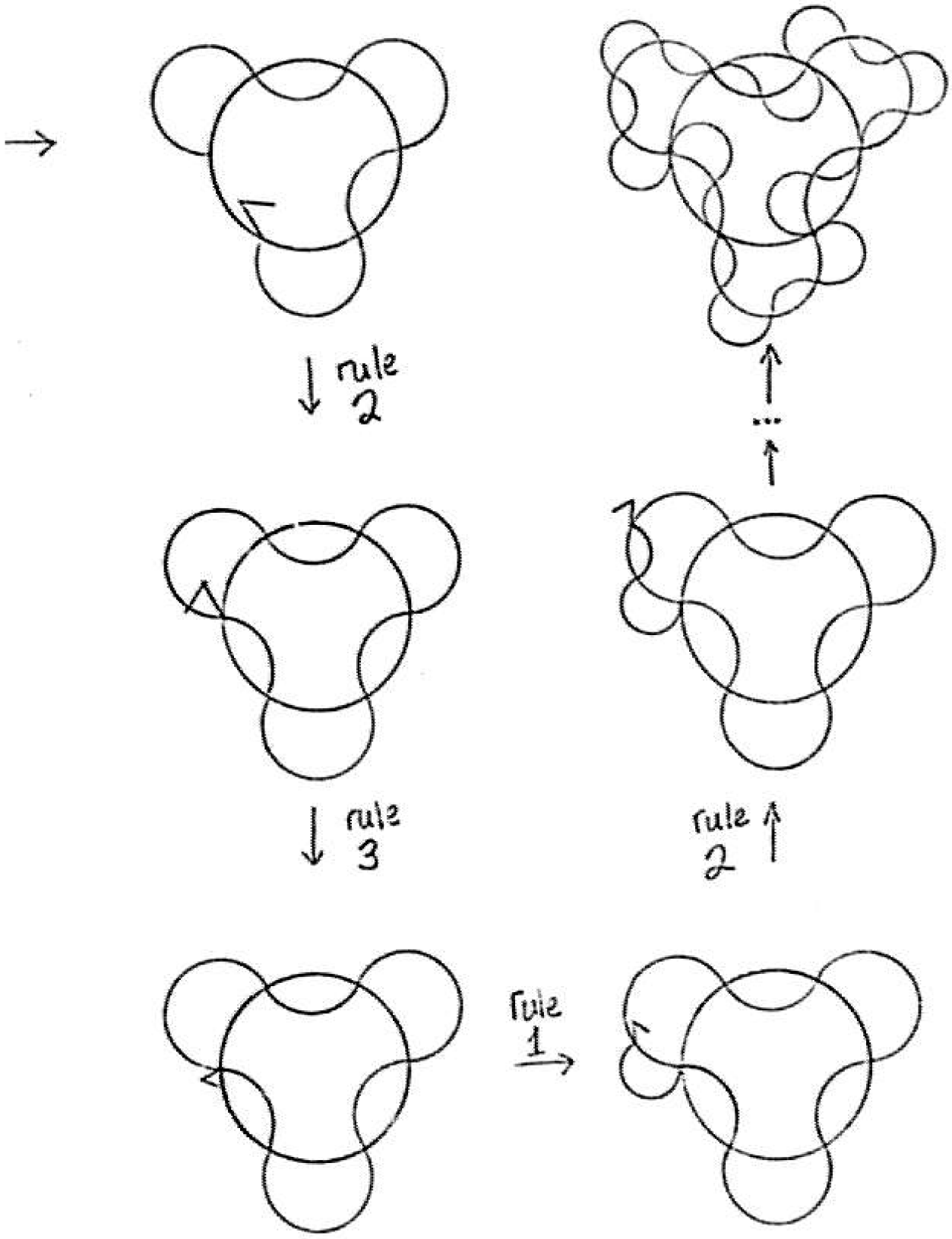


Figure 31b. Generation of shapes used in Triad.

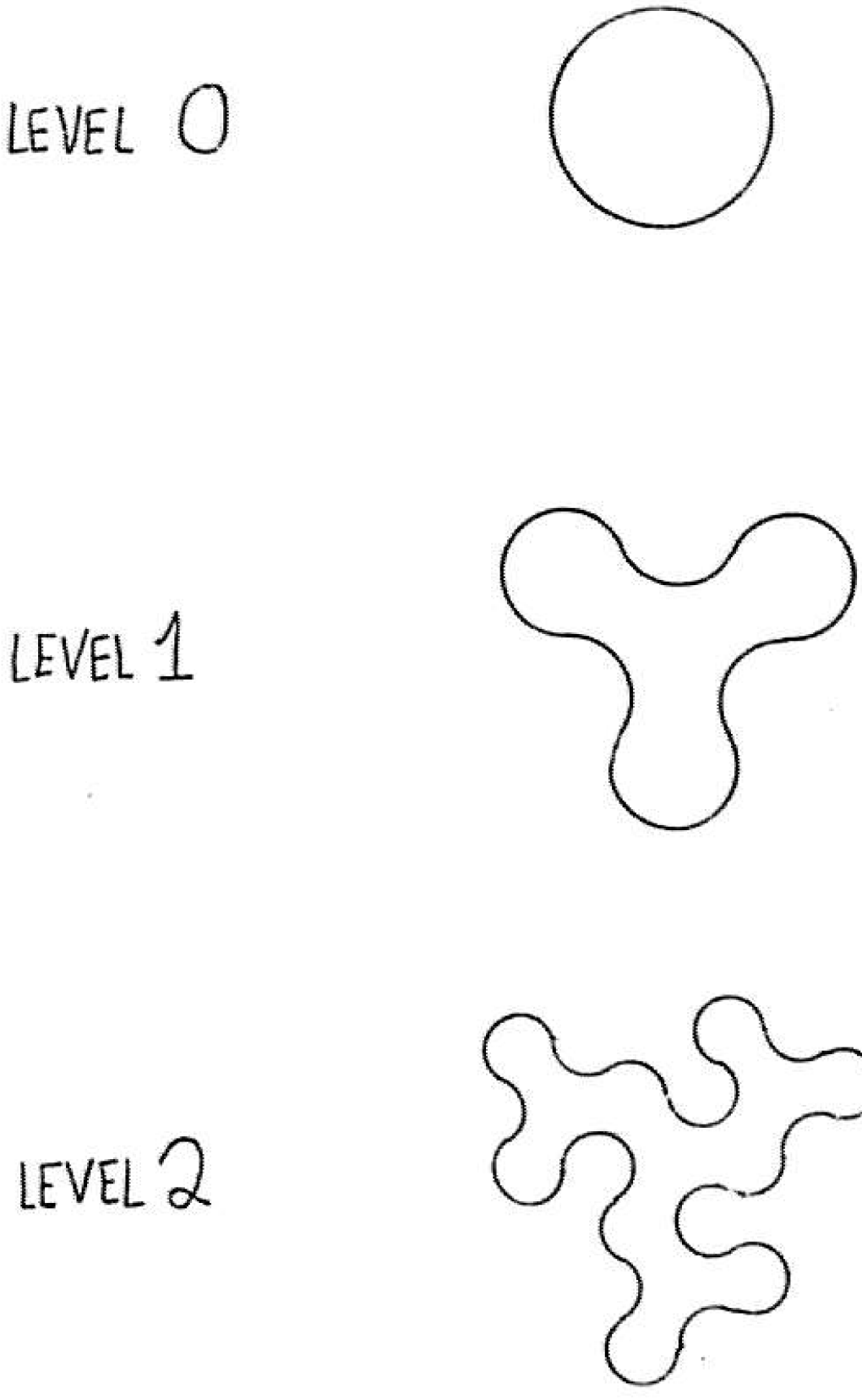


Figure 31c. Level assignments to terminals in shape used in Triad.

be added and the marker to be moved ahead. Rule 2 is applicable when the marker is on the concave side of the next terminal arc. Application of rule 2 causes two terminals (i.e., a 120 degree arc) to be added and the marker to be moved ahead. Rules 3 and 4 are applicable to the initial shape and after the generation of each level of closed curve of terminals. Application of rule 3 causes the marker to be reduced in size and turned, thus forcing the generation to be continued for another level. Application of rule 4 erases the marker, thereby terminating the generation and adding a shape to the language.

3.1.1.2 Selection rule

Painting requires a small class of shapes that are not beyond its techniques for representation. Because a shape grammar can define a language containing a potentially infinite number of shapes ranging from the simple to the very (infinitely) complex, a mechanism (selection rule) is required to select shapes in the language for painting. The concept of level provides the basis of this mechanism and also for the painting rules discussed in the next section.

The level of a terminal in a shape is analogous to the depth of a constituent in a sentence defined by a context free phrase structure grammar. Level assignments are made to terminals during the generation of a shape using these rules:

(1) The terminals in the initial shape are assigned level 0.

(2) If a shape rule is applied and the highest level assigned to any terminal used in the left side of the rule is N, then each terminal added by the application of the shape rule is assigned level N+1.

(3) No other level assignments are made.

The assignment of levels to the terminals in the example is shown in Figure 31c. Notice that a new level is effectively begun each time rule 3 is applied.

A selection rule is an integer which specifies the minimum level required and the maximum level allowed in a shape in the language for it to be a member of the class to be painted. The selection rule determines the depth of generation of the shape grammar required to produce a shape to be painted. Theoretically, several shapes in the language defined by a shape grammar could have the same maximum level. A generative specification defines a class of paintings. However, in every generative specification discussed here (and ever investigated) the language defined by the shape grammar contains only one distinct shape with any given maximum level. Thus, the selection rule normally specifies one shape to be painted from the language and each generative specification normally specifies exactly one painting.

The selection rule in the example, 2, specifies exactly the shape generated in Figure 31b.

3.1.1.3 Painting rules

Painting rules indicate how the areas contained in a shape are to be painted. In applying the painting rules, the generated shape is treated as if it were a Venn diagram as in naive set theory. The terminals of each level in a shape are taken as the outline of a set in the Venn diagram. Levels 0, 1, 2, ... n are said to define sets L_0, L_1, L_2, ... L_n respectively, where n is the selection rule. A painting rule has two sides separated by a double arrow (=>). The left side of a painting rule defines a set using the sets determined by level assignment and the usual set operators, e.g. union, intersection, and complementation. The sets defined by the left sides of the painting rule must partition the Venn diagram. The right side of a painting rule is a rectangle painted in the manner the set defined by the left side of the rule is to be painted. The rectangle gives implicitly medium, color, texture, edge definition, etc. For convenience, the rectangle contains the name of the color the area is to be painted instead of a sample of the painted canvas. It is assumed that acrylic paint is used and the areas are painted flat and have a hard edge. Because the left sides of the painting rules form a partition, every area of the shape is painted in exactly one way. The set notation enables the specification of how areas are to be painted to be independent of the actual shapes of the areas. Notice that any level in a shape may be ignored by excluding the corrsponding set from the left sides of the rules.

The painting rules in Figure 31a in the generative specification for Triad are an example of a painting rule schema that has been frequently used. For a

selection rule of n, there are in general n+1 basic "sets" defined by level assignment and, following this schema, n+2 different painting rules of the form:

L_n => $color_1$

$L_{n-1} \cap \sim(L_n)$ => $color_2$

$L_{n-2} \cap \sim(L_{n-1} \cup L_n)$ => $color_3$

. . .

$L_1 \cap \sim(L_2 \cup \ldots \cup L_{n-1} \cup L_n)$ => $color_n$

$L_0 \cap \sim(L_1 \cup L_2 \cup \ldots \cup L_{n-1} \cup L_n)$ => $color_{n+1}$

$\sim(L_0 \cup L_1 \cup L_2 \cup \ldots \cup L_{n-1} \cup L_n)$ => $color_{n+2}$

Using these painting rules, the color an area is painted depends only on the highest level of a terminal boundary surrounding it. The area enclosed by set L_n is painted $color_1$. The area enclosed by set L_{n-1} but not by set L_n is painted $color_2$, etc. The perceptual effect of painting rules following this schema is to make the painting appear as if opaque versions of the areas bounded by the successive levels were placed on top of each other.

A second painting rule schema has been used. This schema is more difficult to show for a selection rule of n and so will be shown for a selection rule of 2:

$L_0 \cap L_1 \cap L_2$ => $color_1$

$(L_0 \cap L_1 \cap \sim L_2) \cup (L_0 \cap \sim L_1 \cap L_2) \cup (\sim L_0 \cap L_1 \cap L_2)$ => $color_2$

$(L_0 \cap \sim L_1 \cap \sim L_2) \cup (\sim L_0 \cap L_1 \cap \sim L_2) \cup (\sim L_0 \cap \sim L_1 \cap L_2)$ => $color_3$

$\sim(L_0 \cup L_1 \cup L_2)$ => $color_4$

The effect of these painting rules is to count set overlaps. The area enclosed by all three sets is painted $color_1$. The area enclosed by exactly two of the sets is painted $color_2$, by exactly one of the sets $color_3$, and by none of the sets $color_4$. The perceptual effect of painting rules following this schema can be to make the painting appear as if identically tinted, transparent versions of the areas bounded by the successive levels were placed on top of each other.

3.1.1.4 Limiting shape

The limiting shape defines the size and shape of the canvas on which a shape is painted. Traditionally the limiting shape is a single rectangle, but this need not be the case. For example, the limiting shape can be the same as the outline of the shape painted or it can be divided into several parts. The limiting shape is designated by broken lines, and its size is indicated explicitly. The initial shape of the shape grammar in the same scale is located with respect to the limiting shape. The limiting shape in the example is a 32 in. x 32 in. square which contains all of the generated shape. As seen in the example in the next section, the initial shape need not be located within the limiting shape. Informally, the limiting shape acts as a camera viewfinder or a cookie cutter. The limiting shape determines what part of the painted shape is represented on a canvas and in what scale.

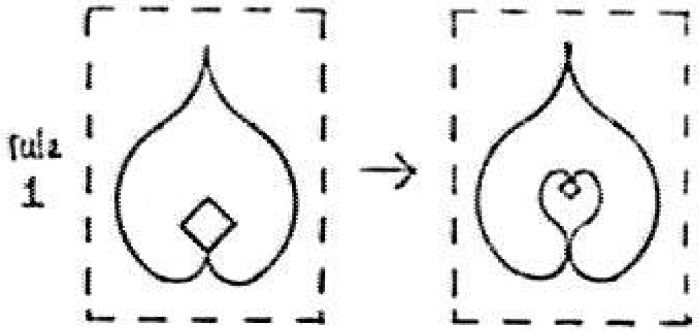
3.1.1.5 Example: Eve

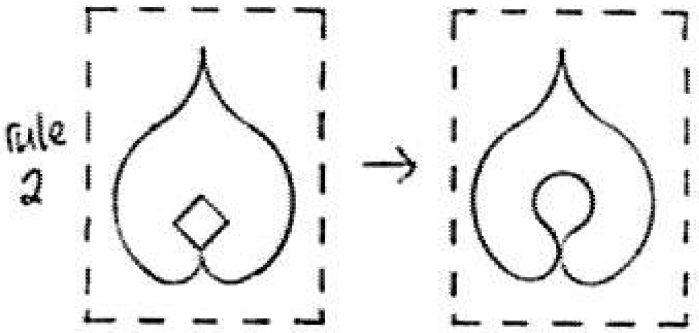
The generative specification of the painting Eve (shown in Figure 30b) is given in Figure 32. The shape grammar in this generative specification contains two terminals and a marker. The first terminal can be decomposed into ten 60 degree circle arcs, six of radius R and four of radius R/3. The second terminal can be decomposed into seven 60 degree arcs, all of the same radius. Both of the shape rules have the same left side. The initial shape consists of one terminal and one marker. The shape grammar is similar to the shape grammar for inscribed squares in Figure 1. The shape in the language specified by the selection rule in the generative specification is generated by applying rule 1 four times and rule 2 once. Both rules are applicable under identical circumstances. The effect of applying the first rule is to add an instance of the first terminal and to move the marker and shrink it by a factor of three, thereby forcing the continuation of the generation process. The effect of applying the second rule is to add an instance of the second terminal and to erase the marker, thereby halting the generation process and producing a shape in the language. The terminal added by each successive application of a rule is assigned a new level. The effect of the painting rules is to paint the areas enclosed by alternate levels of terminals the same color. These painting rules can be considered a special case of each of the two schemata described previously. The painting rules take advantage of the fact that each area enclosed by the terminal of a given level is completely contained in the area enclosed by the terminal of each lower level. The limiting shape in the generative specification is a rectangle that does not completely contain the initial shape.

SHAPE GRAMMAR SG13

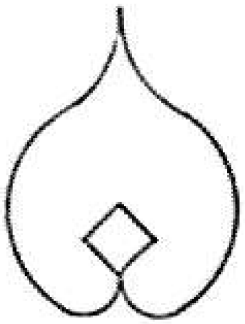
$V_T = \{$  $\}$ $V_M = \{\square\}$

R contains





I is



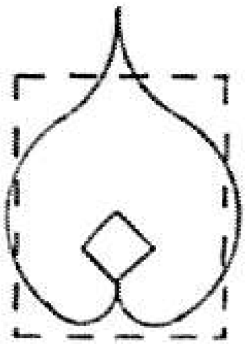
SELECTION RULE : 5

PAINTING RULES

$(L_0 \cap \sim L_1) \cup (L_2 \cap \sim L_3) \cup (L_4 \cap \sim L_5) \Rightarrow$ WHITE

$(\sim L_0) \cup (L_1 \cap \sim L_2) \cup (L_3 \cap \sim L_4) \Rightarrow$ BLUE

LIMITING SHAPE



12 inches

Figure 32. Generative specification for Eve.

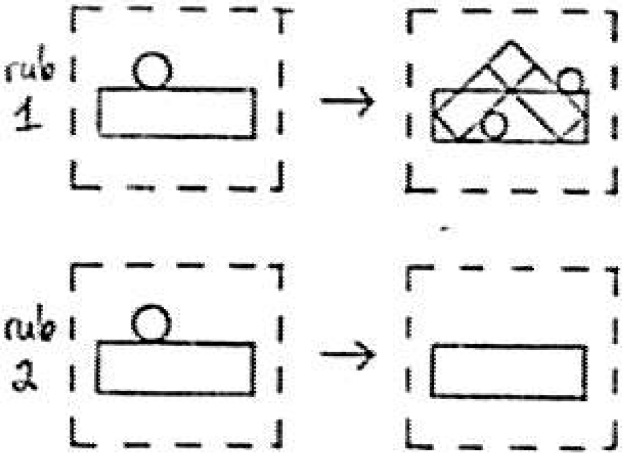
3.1.1.6 Example: Yellow cross

The generative specification of the painting Yellow Cross (see Figure 30c) is given in Figure 33a. The shape grammar in this specification is a parallel shape grammar. There is one terminal, a rectangle, and one marker, a circle. The shape grammar has two shape rules. Note that the right side of the first shape rule contains two overlapped terminals (rectangles) and two markers plus the terminal of the left side of the shape rule. The initial shape contains four markers and four overlapped terminals that form a diamond with a square hole. Recall that in the generation using a parallel shape grammar, when a shape rule is applied, it is applied everywhere it is applicable. Both rule 1 and rule 2 are applicable under identical circumstances. Application of rule 1 adds a new level of terminals and forces the continuation of the generation; application of rule 2 erases the markers and halts the generation. In the example, rule 1 is applied 3 times before the generation is halted with an application of rule 2. The terminals added with each parallel rule application are assigned a new level. The outline of the terminals added by each rule application and assigned each level is shown in Figure 33b. The initial shape (level 0) is composed of four terminals (rectangles). Eight terminals are added in the first parallel application of rule 1 (level 1), sixteen in the second application (level 2), and thirty two in the third application (level 3). The painting rules follow the first schema given in Section 3.1.1.3 except the set L_0 is ignored. Because the set L_0 is not mentioned in the painting rules, the terminal shapes assigned level 0 (i.e. the initial shape) do not appear in the final painting. The limiting shape is a square that wholly contains the generated shape.

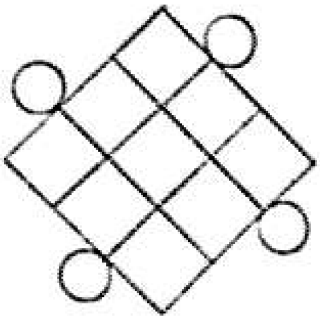
SHAPE GRAMMAR PSG 14

$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains



I is



SELECTION RULE : 3

PAINTING RULES

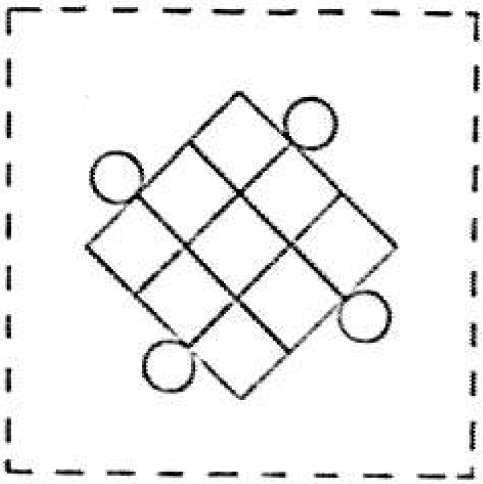
$L_3 \Rightarrow$ YELLOW

$L_2 \cap \sim L_3 \Rightarrow$ RED

$L_1 \cap \sim (L_2 \cup L_3) \Rightarrow$ BLUE

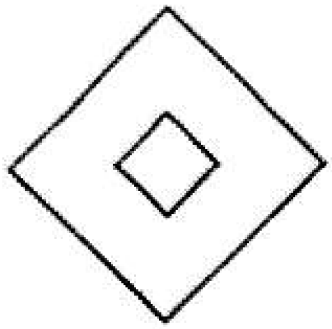
$\sim (L_1 \cup L_2 \cup L_3) \Rightarrow$ WHITE

LIMITING SHAPE

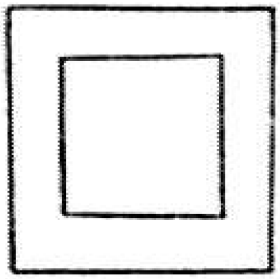


12 inches

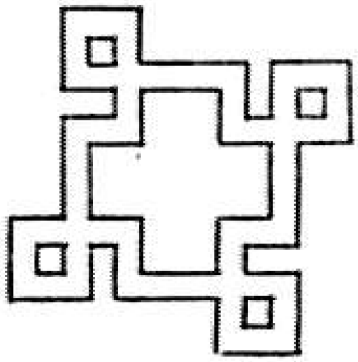
Figure 35a. Generative specification for Yellow Cross.



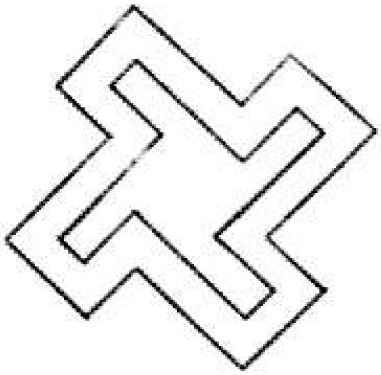
level 0



level 1



level 3



level 2

Figure 33b. Level assignments to terminals in shape used in Yellow Cross.

3.1.2 Computer implementation

A computer program that implements generative specifications has been written in SAIL [VanLehn 1973] and runs on the PDP-10 at the Stanford Artificial Intelligence Laboratory.

The program allows generative specifications for rectilinear paintings to be defined interactively using a keyboard and Data Disc or III display. Once the shape grammar and selection rule of a generative specification are defined, the program genorates and displays the line drawing of the specified shape. From the line drawing, an image of the painting itself can be displayed via a video synthesizer in shades of grey (or green) on the Data Disc displays or in full color on the color television connected to the PDP-10. Hard copy of the line drawing or a low contast shades of gray version of the painting can be obtained using the Xerox Graphics Printer (XGP).

The general structure of the program is diagrammed in Figure 34. The program has two monitor states, which are indicated in the diagram by rectangles with curved corners. The shape grammar is constructed in the first monitor state. When the program is in this state, the user is prompted by an "*". Operations to be performed by the program are indicated by typing a single character and then typing any additional information requested by the program. Details of the commands available to construct the shape grammar are given in Section 3.2.2.1. After a shape qrammar has been defined, "G" is typed and in response to the request by the program, the level of generation desired (i.e., the selection rule) is

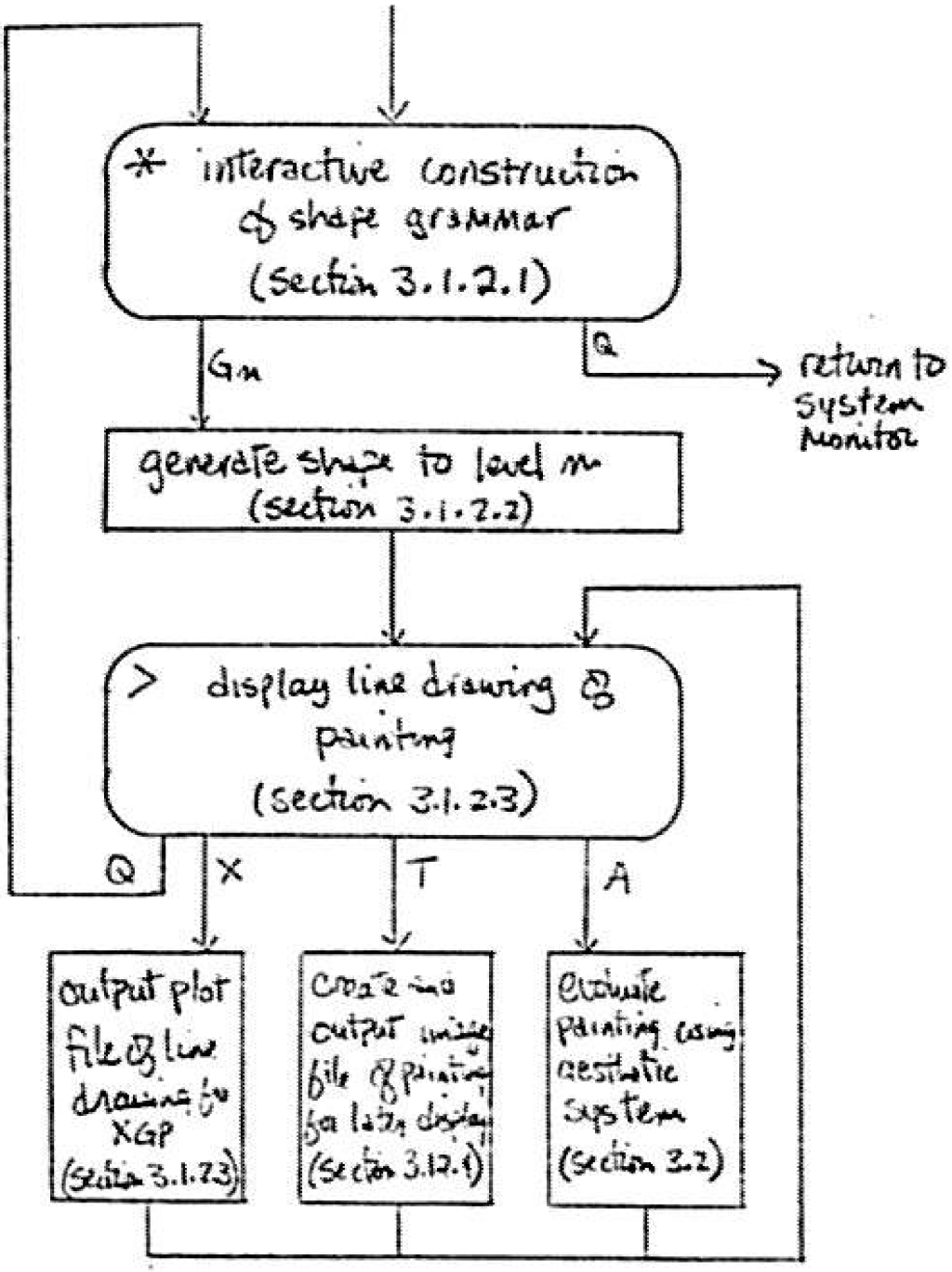


Figure 34. General structure of program for defining generative specifications.

typed. The specified shape is generated (Section 3.1.2.2) and the program enters the second monitor state. When the program is in this state (Section 3.2.2.3), the user is prompted by a ">". From this monitor state, different levels of the generated state can be displayed, an output file for the XGP can be prepared, and a bit file image of the painting can be prepared for color display (Sections 3.1.2.4 and 3.1.2.5). Additionally, by typing "A", the painting can be evaluated using the aesthetic system developed in Section 3.2.5. The aesthetics program itself is described in Section 3.2.6.

3.1.2.1 Interactive definition of shape grammar

A shape grammar can be defined using the program only if it meets these requirements:

(1) There is only one terminal which is a closed shape composed of straight lines.

(2) There is one marker.

(3) There are two shape rules. The left side of each shape rule contains one terminal and marker. The right side of the first shape rule contains the terminal in the left side plus some number of additional terminals and markers. The right side of the second shape rule contains just the terminal in the left side of the rule and no markers.

(4) The initial shape contains an equal number of terminals and markers.

(5) The shape grammar is a parallel shape grammar.

While this is a fairly severe restriction on allowable shape grammars, the class of shape grammars that fulfill these requirements seems sufficiently large and rich in shape grammars of interest for generative specifications. After a year of running the program and after the definition and generation of hundreds of paintings, this class of shape grammars is far from exhausted. The shape grammar in the generative specification shown in Figure 33a of the painting Yellow Cross is a member of this class and is used as an example in this section.

The shape grammar in the generative specification is defined by interactively constructing three shapes - the terminal shape, the rule shape, and the initial shape. Shown in Figure 35 are (1) parts of the shape grammar of the example, (2) the corresponding terminal shape, rule shape, and initial shape, and (3) the internal representation of these shapes.

The commands available for constructing these shapes are shown in the table in Figure 36. Whenever the program is in this monitor state, the current version of one of the three shape is displayed. The shape that is displayed is the shape that the commands affect. The user can switch to working on the terminal shape, the rule shape, or the initial shape by typing "α", "β", or "λ" respectively.

The terminal shape is the terminal of the shape grammar (i.e., the shape in V_t) and is composed of straight lines. At the outset, the terminal shape consists of a single horizontal line with endpoints at coordinates (+200,0) and (-200,0). For all display purposes, the screen is considered a 1024 x 1024 rectangular coordinate system with the origin in the center. Commands are available to add and delete lines. The lines are numbered. One of these lines is the current line and it is this

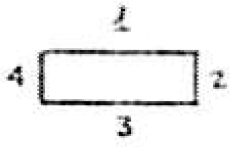
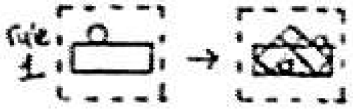
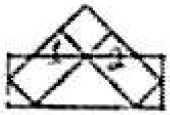
SHAPE GRAMMAR PART	SHAPES DISPLAYED BY PROGRAM	INTERNAL REPRESENTATION
$V_T = \{$ ▭ $\}$	 TERMINAL SHAPE (with lines numbered)	vertex | X | Y 1 | -300 | 100 2 | 300 | 100 3 | 300 | -100 4 | -300 | -100
Rule 1: 	 RULE SHAPE (with shapes numbered)	Shape | XDIS | YDIS | SCALE | θ | INVERT 1 | -100 | 100 | .7071 | 225° | 1 2 | 100 | 100 | .7071 | 315° | 1
I is 	 INITIAL SHAPE (with shapes numbered)	Shape | XDIS | YDIS | SCALE | θ | INVERT 1 | -106 | 159 | .75 | 45° | 0 2 | 106 | 159 | .75 | 315° | 0 3 | 106 | -53 | .75 | 225° | 0 4 | -106 | -53 | .75 | 135° | 0

Figure 35. Computer representation of shape grammar for Yellow Cross.

COMMAND	INFORMATION REQUESTED	DESCRIPTION
α, Z		switch to and display terminal shape
β, X		switch to and display rule shape
λ, V		switch to and display initial shape
C	line/shape no.	make the indicated line/shape the Current line/shape
D		Delete the current line/shape and renumber if necessary
E	factor	Enlarge current line/shape by indicated factor, where factor is a real number. If the letter Q precedes the number, the square root of the number is used.
F (terminal shape only)	fasten point to line no. point	Fasten - move the current line horizontally and/or vertically so that the indicated endpoint of the current line coincides with the indicated endpoint of the indicated line. Endpoints are indicated by ← (leftmost), → (rightmost), ↑ (upper), ↓ (lower), or ↔ (midpoint).
F (rule shape or initial shape only)	fasten line no. point to shape no. line no. point	Fasten - move the current shape horizontally and/or vertically so that the indicated endpoint of the indicated line of the current shape coincides with the indicated endpoint of the indicated line of the indicated shape. Endpoints are indicated as in Fasten for terminal shape.
G	selection rule	use constructed shape grammar to Generate a shape to the indicated level and enter second monitor level (see Figure 37)
I (rule shape or initial shape only)		Invert the current shape
M		Mark (display) the numbers assigned to lines/shapes
N		create a New copy of the current line/shape
O		reinitialize - return terminal, rule, and initial shapes to Original state

COMMAND	INFORMATION REQUESTED	DESCRIPTION
Q		Quit - return to system monitor
R	file name	Read new shape grammar from disk
S	factor	Shrink current line/shape by indicated factor, where factor is a real number. If the letter Q precedes the number, the square root of the number is used.
T	angle	Turn the current line/shape the indicated no. of degrees counterclockwise
W	file name	Write current shape grammar on disk
Y		tYpe the coordinates/parameters of the displayed lines/shapes
→, K	distance	move the current line/shape the indicated distance to the right
←, J	distance	" to the left
↑, :	distance	" upwards
↓, :	distance	" downwards

Figure 36. Summary of commands available for constructing shape grammars.

line that is altered by the commands to enlarge, shrink, move, rotate, and fasten (see table in Figure 36). Any line can be specified as the current line by using the "C" command. During the construction of the terminal shape, the terminal shape is represented as a 4 x t two-dimensional integer array where t is the maximum number of lines allowed in the terminal shape. Each line is represented by four numbers - the x and y coordinates of the endpoints. For the generation of a shape (see next section), this representation is changed to take advantage of the requirement that the terminal shape ultimately must be closed. During shape generation, the terminal shape is represented by the sequence of coordinates of vertices encountered in a counter-clockwise trace around the shape, beginning with the first vertex of the first line.

The rule shape represents the terminals added in the right side of the first shape rule in the shape grammar and is composed of instances of the terminal shape. The terminal on the left side of the first shape rule is always a single instance of the terminal shape. The rule shape is displayed superimposed on the terminal shape. This forms the terminals on the right side of the first shape rule. At the outset, the rule shape consists of a single instance of the terminal shape. Commands are available for adding and deleting instances of the terminal shape. The instances of the terminal shape are numbered. One of these instances is the current shape and it is this instance that is altered by the commands to enlarge, shrink, move, rotate, invert, and fasten (see table in Figure 36). Any instance can be specified as the current shape using the choose command.

The initial shape represents, as the name implies, the initial shape of the

shape grammar and is also composed of instances of the terminal shape. The commands for manipulating the terminals in the initial shape are just about identical to the commands for manipulating the terminals in the rule shape.

For both the rule shape and the initial shape, each instance of the terminal shape is represented internally by five parameters: the x displacement, the y displacement, the scale s, the counter-clockwise rotation in degrees θ, and mirror image m. These parameters determine the coordinates of the particular instance of the terminal shape. If (OLDX, OLDY) are the coordinates of a vertex in the terminal shape and XDIS, YDIS, SCALE, THETA, and INVERT are the parameters of an instance of the terminal shape in the rule shape, the new coordinates (NEWX, NEWY) of the vertex in the rule shape are computed by :

```
IF INVERT ≠ 0 THEN OLDX ← -OLDX;

IF THETA ≠ 0
THEN BEGIN
     TEMP ← OLDX * COS(THETA) - OLDY * SIN(THETA);
     OLDY ← OLDX * SIN(THETA) + OLDY * COS(THETA);
     OLDX ← TEMP
     END;

NEWX ← SCALE * OLDX + XDIS;

NEWY ← SCALE * OLDY + YDIS;
```

This defines the transformations that are used but is not the precise algorithm, e.g., cos and sin are precomputed and stored in a table and the original values of OLDX and OLDY are not destroyed. Notice that the order the transformations are applied

makes a difference in the effect of specific parameters. For a good discussion of two-dimensional (and three-dimensional) transformations see [Newman and Sproull 1973].

The five parameters that represent each instance of the terminal shape in either the rule shape or initial shape can be considered to define a new coordinate system. The transformations just presented give the mapping from the original screen coordinate system to this new coordinate system. If the numbers of the shapes are displayed (see the "M" command in Figure 36), the number of each shape is displayed at the origin of the new coordinate system defined by the shape, i.e. at (XDIS, YDIS) in the screen coordinate system. If the operation of rotation ("turn") or inversion is specified for a shape, the operation is performed relative to the coordinate system defined by the shape, e.g. the shape is rotated about its own origin rather than the origin of the screen coordinate system. On the other hand, the operations of translation (x and y displacement) are performed relative to the screen coordinate system. These conventions may appear to be arbitrary, but in practice they seem most natural.

Once a shape grammar (i.e. terminal shape, rule shape, and initial shape) has been constructed, it can be saved on the disk using the write command and subsequently reinstated using the read command (see Figure 36). To generate a shape using a constructed shape grammar, "G" is typed and then the desired level of generation (selection rule) is specified.

3.1.2.2 Shape generation

A shape grammar of the restricted type allowable for the program always has two shape rules. Application of the first shape rule during a generation always results in the addition of a new level of terminals, the addition of new markers, and the continuation of the generation process. Application of the second shape rule always results in the addition of no new terminals, the erasure of all markers, the termination of the generation process, and a shape in the language. If the selection rule of a generative specification is n, the shape to be painted is the shape in the language generated by the shape grammar which contains terminals assigned a maximum level of n. Using a shape grammar of this restricted type, a shape containing terminals assigned a maximum level of n always can be generated by recursively applying the first shape rule to the initial shape n times and then applying the second shape rule.

For the program, the shape grammar is defined completely by the three shapes - terminal shape, rule shape, and initial shape - described in the previous section. The terminal in the left side of the first shape rule consists of a single instance of the terminal shape. The parameters of this instance of the terminal shape are implicitly x=0, y=0, s=1, θ=0, and m=0. The terminals in the right side of the first shape rule are given by the rule shape. The relative locations of the markers in the first shape rule are given implicitly by the specific parameters of the rule shape. The terminals of the right and left sides of the second shape rule are identical and consist of single instances of the terminal shape. The terminals of

the initial shape of the shape grammar are specified by the initial shape constructed using the program. The markers in the initial shape are given implicitly by the specific parameters.

The program generates a shape by recursively applying the transformations specified by the rule shape to the initial shape. The shape that is generated consists of multiple instances of the terminal shape.

Assume there are r instances of the terminal shape in the rule shape, t instances of the terminal shape in the initial shape, and the selection rule (level to be generated) is n. Level 0 of the generated shape is the initial shape and so consists of t instances of the terminal shape. Level 1 is generated by applying the shape rule to each instance of the terminal shape and so consists of rt instances of the terminal shape. Level 2 is generated by applying the shape rule to each instance of the terminal shape in level 1 and so consists of r^2t instances of the terminal shape, etc. Level n consists of r^nt instances of the terminal shape. Thus the generated shape consists of $t + rt + r^2t + \ldots + r^nt$ instances of the terminal shape. This is a geometric series the sum of which is equal to $(n+1)t$ if $r=1$ and $t(1-r^{n-1})/(1-r)$ for $r>1$. In the example (see Figures 30c, 33, and 35), r=2, t=4, and n=3. Level 0 contains four instances of the terminal shape, level 1 contains eight, level 2 sixteen, and level 3 thirty two. The generated shape contains sixty instances of the terminal shape.

The terminal shape must be a closed shape. After its construction is completed, the terminal shape is represented as a sequence of (x,y) pairs where each pair gives the coordinates of one of the vertices. Each instance of the

terminal shape in the generated shape is represented internally by the five parameters - x, y, θ, s, and m - described in the previous section. The terminals of level i are generated from the terminals of level i-1 by applying each of the r sets of transformations in the rule shape to each of the $r^{i-1}t$ instances of the terminal shape of level i-1 to produce $r^i t$ new instances of the terminal shape in level i. For each terminal in level i-1, r new terminals of level i are generated. If RXDIS, RYDIS, RTHETA, RSCALE, and RINVERT are copies of the parameters of one of the r terminal shapes in the rule shape and OLDXDIS, OLDYDIS, OLDTHETA, OLDSCALE, and OLDINVERT are the parameters of one of the $r^{i-1}t$ shapes at level i-1, then the parameters NEWXDIS, NEWYDIS, NEWTHETA, NEWSCALE, and NEWINVERT of the resulting terminal shape of level i are given by the algorithm :

```
IF OLDINVERT
THEN BEGIN  RTHETA ← -RTHETA;  RXDIS ← -RXDIS  END;

NEWXDIS ← OLDXDIS + OLDSCALE * (RXDIS * COS(OLDTHETA)
                              - RYDIS * SIN(OLDTHETA));

NEWYDIS ← OLDYDIS + OLDSCALE * (RYDIS * COS(OLDTHETA)
                              + RXDIS * SIN(OLDTHETA));

NEWTHETA ← OLDTHETA + RTHETA;

NEWSCALE ← OLDSCALE * RSCALE;

NEWINVERT ← (OLDINVERT + RINVERT) MOD 2;
```

Thus the generated shape consists of multiple instances of the terminal shape where each instance is specified by five parameters. To obtain the coordinates of the vertices of a particular instance of the terminal shape, the algorithm given in the previous section is used.

The program allows the generation of much more intricate shapes than I had patience to generate by hand. Shapes containing over 5000 lines have been generated using 10 - 15 seconds of CPU time. Flicker is not a problem if a Data Disc display (a raster-type display) is used. A few examples of the types of shapes generated are given in Sections 3.1.2.7 - 3.1.2.9.

3.1.2.3 Display of line drawing

After the line drawing is generated, it is displayed and the program enters the second monitor state. The commands available in this monitor state are shown in Figure 37.

If the user wants to change the sh pe grammar and generate a different line drawing, typing "Q" returns the program to the shape grammar construction monitor. The display command "D" enables one level or a series of levels to be displayed by itself. The commands in this monitor state operate on exactly the levels that are displayed when the commands are typed. Recall that in the painting Yellow Cross used as an example, level 0 (the initial shape) is ignored and just levels 1 thru 3 are used. To obtain the shape used in this painting, the shape grammar of Figure 35 is constructed, used to generate a shape to level 3, and the shape is displayed from level 1 to level 3.

To specify a limiting shape, "B" is typed. For the program, the limiting shape is always a rectangle located around and outside the generated shape. After "B" is

COMMAND	INFORMATION REQUESTED	DESCRIPTION
A		run the Aesthetics program to evaluate the generative specification and painting in terms of the programmed aesthetic system (see Section 3.2)
B	distance	set the Border of the limiting shape. The limiting shape is always a rectangle the indicated distance from the maximum and minimum x and y coordinates of the generated shape.
D	from level to level	Display those consecutive levels of the generated shape between and including the indicated levels
O	x:y aspect ratio file name	Output a plot file of the generated shape for the XGP taking into account the indicated x:y aspect ratio of the XGP
Q		Quit - return to first monitor level (see Figure 36)
S	α (terminal shape), β (rule shape), or λ (initial shape)	Show (display) the indicated shape of the shape grammar without leaving this monitor level
T	x:y aspect ratio file name	prepare and output image file for color or black and white Tv display of the final painting, taking into account the indicated x:y aspect ratio
&	file name	prepare and output image file in standard hand-eye format

Figure 37. Summary of commands available during display of generated shape.

typed, the program asks "BORDER = ". The positive number typed in response by the user is the distance from the minimum and maximum x and y coordinates of the generated shape that the rectangular limiting shape is to be located.

To get a hardcopy drawing of the shape displayed, "0" is typed. This causes a display file to be written on the disk. This file can later be used for printing on the Xerox Graphics Printer (or plotting on the Calcomp). Since the x:y aspect ratio of the XGP is not normally 1 (e.g. squares get drawn as rectangles), provision is made for the user to specify the current XGP aspect ratio and for the program to automatically correct for this.

So far, the line drawing of the generated shape has been displayed. This corresponds to the implementation of the shape grammar and selection rule. In the generative specification, the coloring rules define how the areas contained in the shape are to be painted. This operation in the program is done in two parts, which are described in the next two sections.

3.1.2.4 Transformation of line drawing to image of the painting

To display a colored image of the painting on the color television, an image file is first generated. This file contains three bits for each point in the picture to be displayed. The octal number assigned to each point indicates the color at the point. Because the color television on which the painting will be displayed is considered to have a 512 x 512 grid, an image file can contain up to 512*512*3 = 768k bits or approximately 21k words.

To create an image file for a displayed line drawing, "T" is typed from the monitor state just described. If no border has been specified (see previous section), the program requests one. Because the line drawing is defined in a 1024 x 1024 coordinate system and the image in a 512 x 512 coordinate system, each coordinate in the line drawing will be divided by two. An array large enough to contain the image of the painting is defined and zeroed. Recall that the terminal shape is required to be a closed shape and that the generated shape is composed of instances of the terminal shape. The terminal shapes in the generated shape are taken one at a time, beginning with the shapes at the lowest level (usually the initial shape) and concluding with the terminal shapes of the highest level. For each shape, all the coordinates contained in the shape are determined. For each of these points, the corresponding three bits in the image array is assigned the value (level_of _shape + 1) MOD 8 independent of the previous content of those bits. At the completion of this process for the shapes of all levels, all of the points contained in the highest level are assigned (highest_level + 1) MOD 8. All of the coordinate points contained in terminal shapes assigned the second highest level but not in the terminal shapes assigned highest level are assigned (highest_level) MOD 8. All of the coordinate points in the third highest level shapes but not in the two highest are assigned (highest_level - 1) MOD 8, etc. The coordinate points contained in none of the generated terminal shapes still have value 0. If more than seven levels of shapes have been generated, a highly unusual occurrence, there will be an overlap in the values assigned. This algorithm implements the first painting rule schema described in Section 3.1.1.3 and used in the examples of

Sections 3.1.1.5 and 3.1.1.6. All generative specifications defined using the program are assumed (required) to have painting rules that follow this schema.

The various devices on which the picture can be displayed have different x:y aspect ratios. Before the image file is generated, the user is asked the x:y aspect ratio of the display device to be used.

The image file created and written on the disk using the "T" command has a format designed for use by the program described in the next section but different than the standard picture file formats in use on the system. By typing "&" instead of "T", an image file in standard hand-eye format is produced. This file can be used by the various system programs for displaying shades-of-gray pictures on the Data Disc displays or printing them on the XGP.

3.1.2.5 Display of painting

A separate program, TVDIS, is used to display the image file constructed for a painting. The commands available to the user of TVDIS are shown in Figure 38.

The program allows an image file to be read from the disk. The file contains an octal number for each coordinate point in the painting. This octal number indicates the number of the color the point is to be displayed. The specific color associated with each number is determined by the user of TVDIS using the "C" command. Colors are specified in terms of red, blue, and green intensity values. The intensity of each color component can range from 0 to 63, so there are 256k

COMMAND	INFORMATION REQUESTED	DESCRIPTION
C	level red, blue, and green values	Change colors for indicated level to indicated values
D		Display color picture
E		Erase picture
G		display shades-of-Grey picture using intensity only
I	file name	Input new picture (image file) to be displayed
Q		Quit - return to system monitor
S		Show current colors for each level

Figure 38. Summary of commands available for display of painting.

different possible colors in theory. The program automatically transforms these red, blue, and green values into the intensity and two color components (X and Y) standardly used in color television broadcasting using the following formulas :

```
INTENSITY ← (RED + BLUE + GREEN) DIV 3
XCOL      ← RED - INTENSITY + 32
YCOL      ← GREEN - INTENSITY + 32
```

Because some of the extreme combinations of red, blue, and green values cause the values assigned to XCOL and YCOL to fall outside of the range 0 to 63, there are somewhat less than 256k different colors allowable. The current color components assigned to the points of each level can be shown using the "S" command.

Display on the color television requires twelve Data Disc channels, a large resource which is normally available only at night. The program uses the the input image file and the color components specified for each level number to construct the twelve Data Disc files which are then displayed.

The painting can also be displayed in shades of grey (or green) on the Data Disc displays. This requires only six Data Disc channels and is more frequently possible.

3.1.2.6 The number of memory words used to represent a generative specification

To review, the process of defining a generative specification and generating and displaying the resulting painting is as follows :

(1) The terminal shape, a closed shape which is composed of straight lines, is constructed.

(2) The rule shape, which is composed of instances of the terminal shape, is constructed.

(3) The initial shape, which is also composed of instances of the terminal shape, is constructed.

(4) The selection rule is specified and the resulting shape is generated.

(5) A sub-sequence of the levels of the generated shape is chosen and displayed (optional).

(6) The border is specified and the image file is generated.

(7) The colors for each of the levels and the background are specified and the painting is displayed.

The shape grammar is specified in steps (1) - (3), the selection rule in step (4), the painting rules in steps (5) and (7), and the limiting shape in step (6).

How many words of memory are used by the program to represent each generative specification? This is important for the computer implementation of the aesthetic system (see Section 3.2.6) and may as well be answered here.

(1) Let NVT be the number of vertices in the terminal shape. The internal representation of the terminal shape uses two words for each vertex (one word each for the x and y coordinates) and one word to specify the number of vertices for a total of 2*NVT+1.

(2) Let NRS be the number of instances of the terminal shape in

the rule shape. Each instance uses five words of memory, one for each parameter. One word is used for NRS. The total number of words used for the rule shape is 5*NRS+1.

(3) Let NIS be the number of instances of the terminal shape in the initial shape. The internal representation is the same as the rule shape. Total for the initial shape is 5*NIS+1.

(4) The selection rule uses one word.

(5) To specify the sequence of levels of the generated shape to be used in the painting, one word (in addition to the selection rule) is used.

(6) The border uses one word.

(7) Let the number of levels of the generated shape to be painted, as specified in (4) and (5), be NLEV. The specification of the color a level is to be painted uses three words, one each for the red, blue, and green components. The number of words used to specify the colors is three for each level of the shape to be painted plus three for the background for a total of 3*NLEV+3.

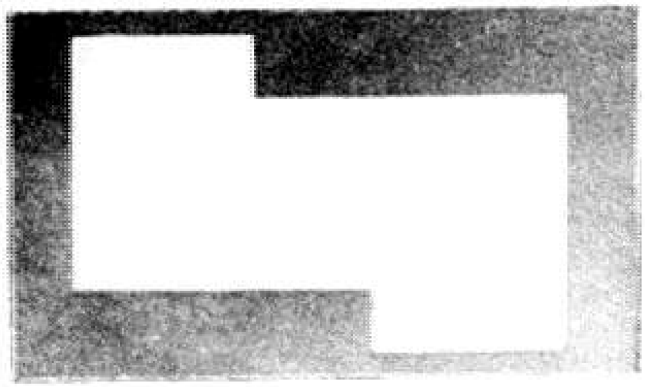
The grand total is 2*NVT + 5*NRS + 5*NIS + 3*NLEV + 9.

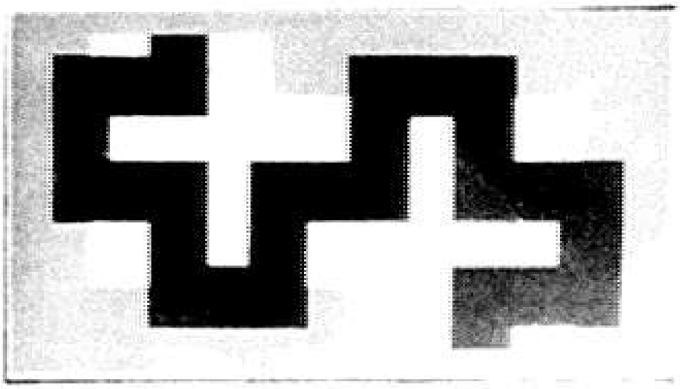
For the painting Yellow Cross used as an example: The number of vertices in the terminal shape (NVT) is 4. The number of instances of the terminal shape in the rule shape (NRS) is 2. The number of instances of the terminal shape in the initial shape (NIS) is 4. The number of levels of the generated shape displayed in the painting (NLEV) is 3. Thus the number of words of memory used to represent this generative specification is 56.

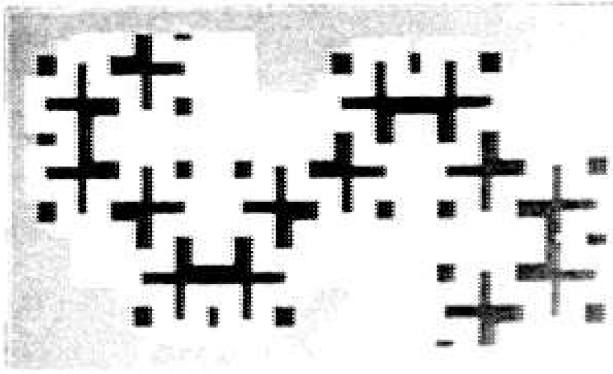
3.1.2.7 Example: Urform

The paintings shown in Figure 39 are defined by the generative specification of Figure 40 using a selection rule of 0, 1, 2, and 3 respectively. These paintings, and the paintings shown in the next two sections, were created using the program; the pictures of the paintings are photographs of the computer display. While it may not be apparent from the black and white photographs in Figure 39, each individual level is colored identically in all paintings in which it occurs. The paintings in Figure 39 are a variation of the paintings used as an example in [Stiny and Gips 1972]

The terminal in the shape grammar is an "L" shape that consists of six straight lines. The right side of the first shape rule contains seven terminals and markers in addition to the terminal present in the left side of the rule. The initial shape contains two terminals and markers. In the construction of the shape grammar using the program, the terminal shape consists of six straight lines, the rule shape of seven instances of the terminal shape, and the initial shape of two instances of the terminal shape. The shape generated using 3 as the selection rule contains 800 instances of the terminal shape.







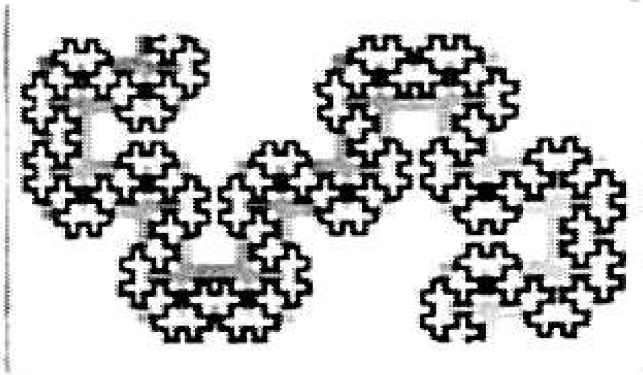
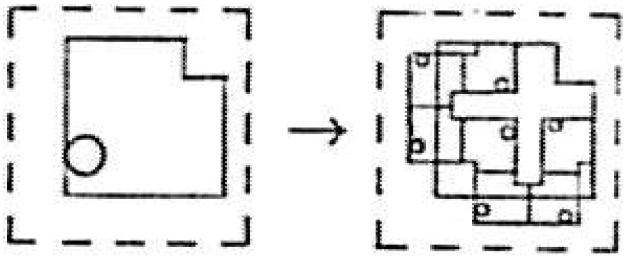


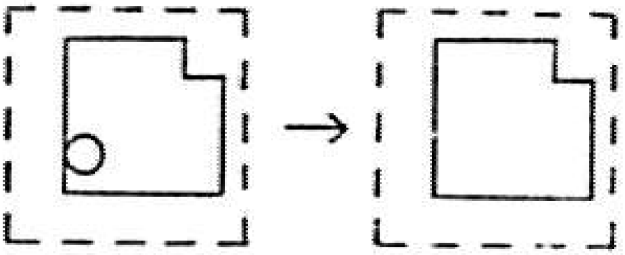
Figure 39. Urform. Colors are black, blue, red, light blue, white (darkest to lightest) in the final painting.

SHAPE GRAMMAR PS#15

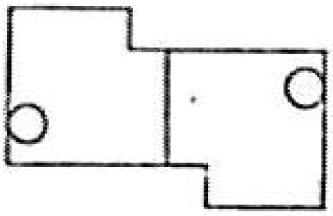
$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains





I is



SELECTION RULE: 3

PAINTING RULES

$L_3 \Rightarrow$ BLUE

$L_2 \cap \sim L_3 \Rightarrow$ LT. BLUE

$L_1 \cap \sim(L_2 \cup L_3) \Rightarrow$ BLUE

$L_0 \cap \sim(L_1 \cup L_2 \cup L_3) \Rightarrow$ WHITE

$\sim(L_0 \cup L_1 \cup L_2 \cup L_3) \Rightarrow$ RED

LIMITING SHAPE

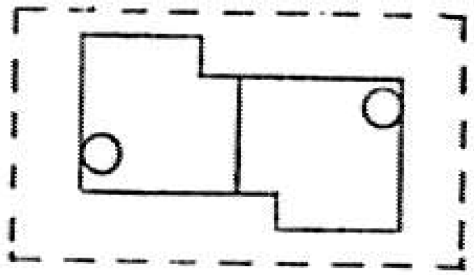


Figure 40. Generative specification for Urform.

3.1.2.8 Example: Star

The paintings shown in Figure 41a are defined using the generative specification of Figure 42 with a selection rule of 0, 1, 2, and 3. Each individual level is colored identically in all paintings in which it occurs, e.g. the background is colored light blue in all the paintings in Figure 41. The terminal in the shape grammar is composed of six straight lines. The right side of the first shape rule contains five terminals and markers in addition to the terminal that occurs in the left side of the shape rule. The initial shape is composed of six overlapping terminals and six markers. In the definition of the shape grammar using the program, the terminal shape is composed of six lines, the rule shape of five instances of the terminal shape and the initial shape of six instances of the terminal shape. The shape generated using a selection rule of 3 contains 936 instances of the terminal shape.

The paintings shown in Figure 41b are defined using variations of the generative specification of Figure 42. These generative specifications are identical to the generative specification of Figure 42 except for the initial shape. In the generative specification for the first of these paintings, the six terminals in the initial shape do not overlap, but are aligned tangentially to form a six pointed star. In the generative specification for the second of these paintings, the initial shape consists of a single instance of the terminal shape.

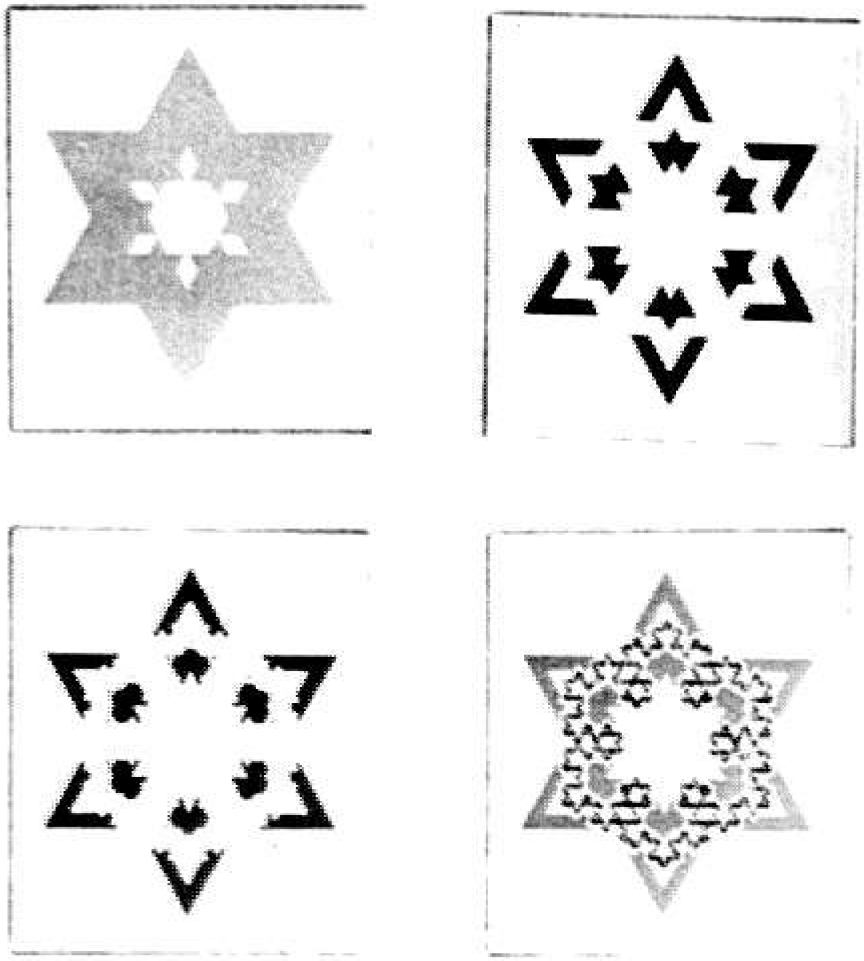


Figure 41a. Star. Colors are dark blue, blue, green, light blue, and white (darkest to lightest) in the final painting.

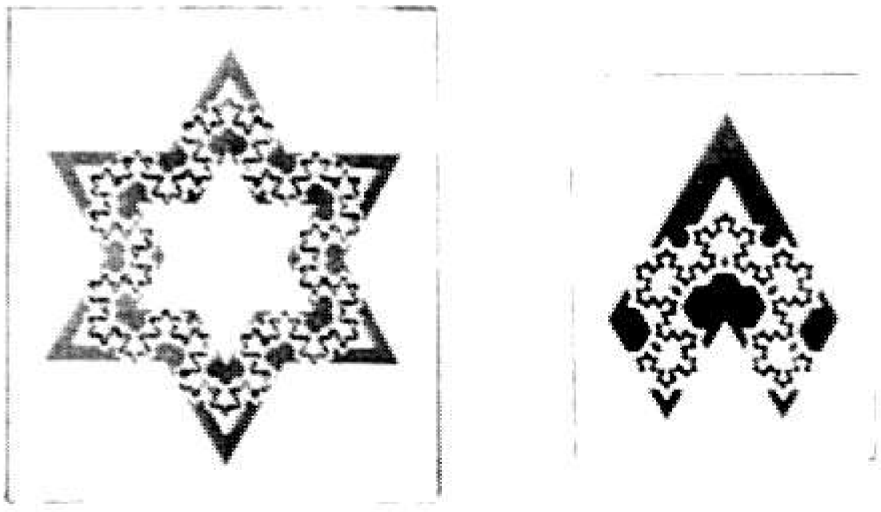


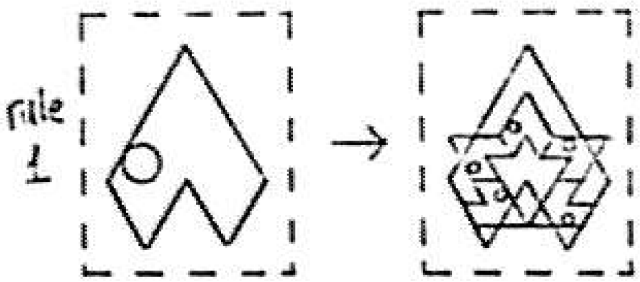
Figure 41b. Variations. Same colors.

Figure 41. Star and variations.

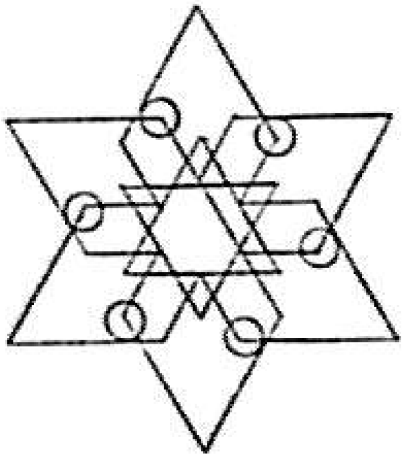
SHAPE GRAMMAR PSG16

$V_T = \{$  $\}$ $V_M = \{$  $\}$

R contains

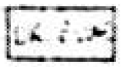


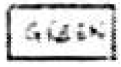
I_{15}

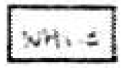


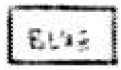
SELECTION RULE: 3

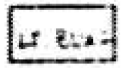
PAINTING RULES

$L_3 \Rightarrow$ 

$L_2 \cap \sim L_3 \Rightarrow$ 

$L_1 \cap \sim (L_2 \cup L_3) \Rightarrow$ 

$L_0 \cap \sim (L_1 \cup L_2 \cup L_3) \Rightarrow$ 

$\sim (L_0 \cup L_1 \cup L_2 \cup L_3) \Rightarrow$ 

LIMITING SHAPE

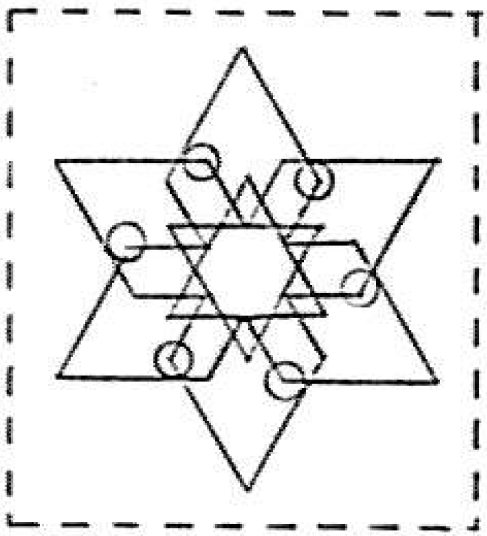


Figure 42. Generative specification for Star.

3.1.2.9 Example: Aleatory

The shape grammars shown previously in the specification of painting have been carefully constructed to insure that the generated terminals of each level are aligned. This enables the details of the shape generated using a relatively high selection rule to be discernible when the shape or painting is displayed. The shape grammars need not be so carefully constructed.

The paintings shown in Figure 43 were generated using a rather arbitrary shape grammar, which is shown in the generative specification in Figure 44. This generative specification defines the most complicated painting in Figure 43. The other paintings can be defined using the same generative specification, but with a selection rule of 1, 2, 3, or 4. Again, each individual level is colored identically in all the paintings in Figure 43 in which it occurs. The terminal in the shape grammar is a rectangle. The right side of the first shape rule contains two terminals and markers in addition to the terminal in the left side of the rule. The initial shape contains one instance of the terminal shape and one marker. Because the set L_0 does not occur in the left side of any of the painting rules, the initial shape is ignored in the paintings.

This concludes the section describing generative specifications and their computer implementation. The use of generative specifications led directly to the investigation of aesthetics reported in the next section.

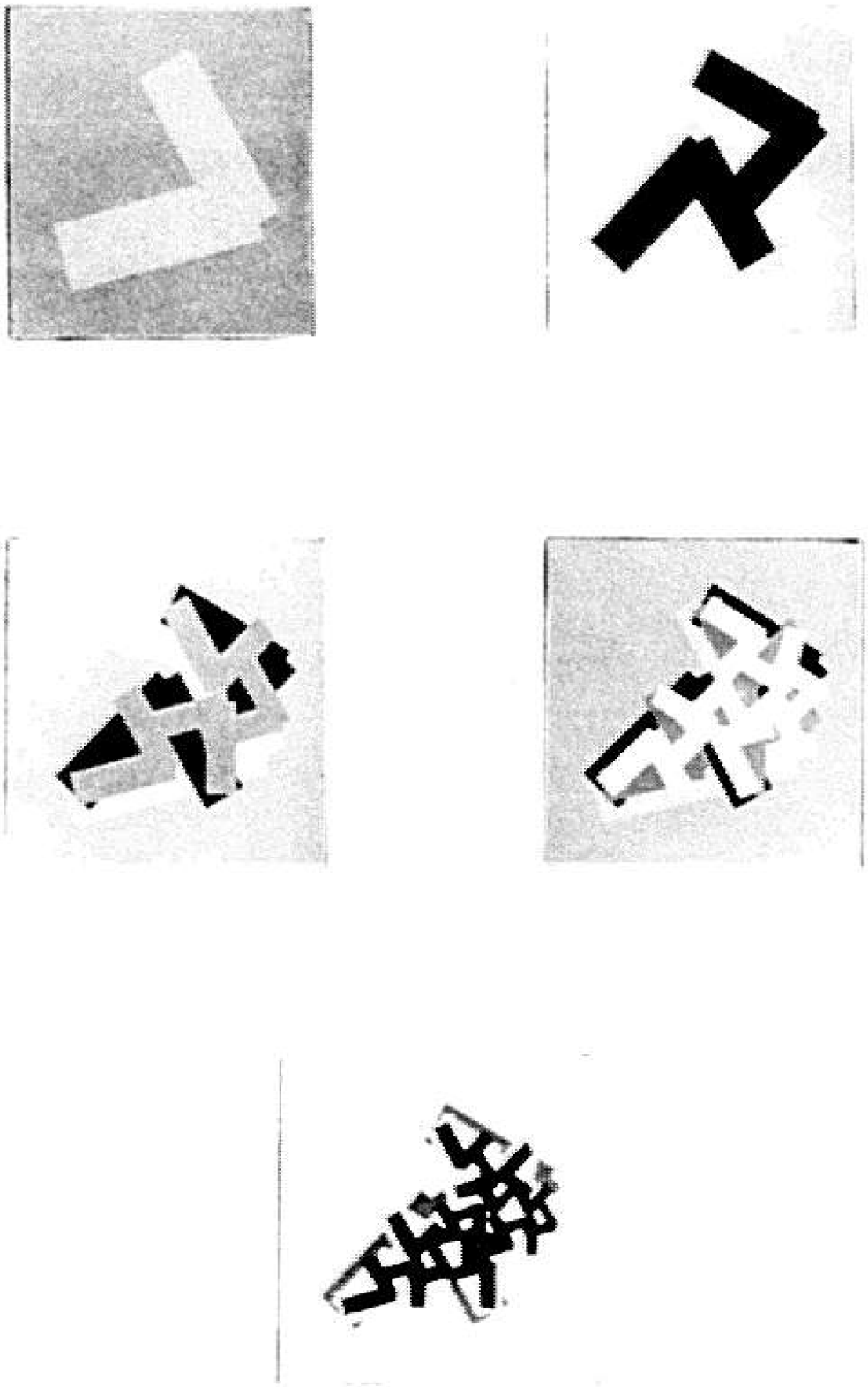
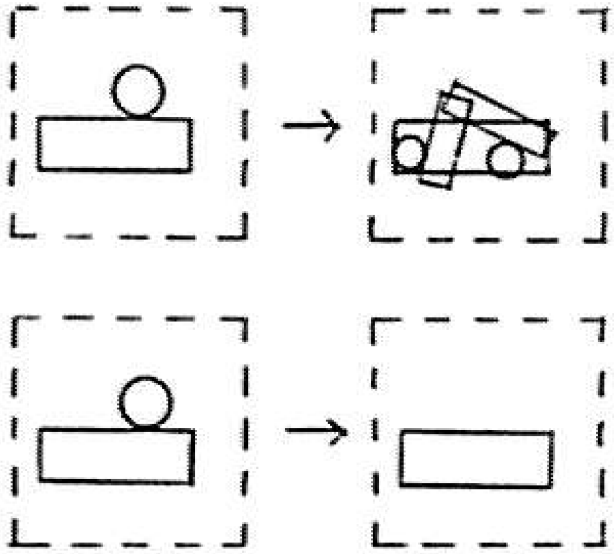


Figure 43. Aleatory. Colors are dark blue, dark green, red, light blue, yellow, white (darkest to lightest) in the final painting.

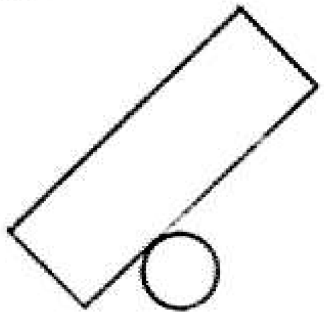
SHAPE GRAMMAR PSG17

$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains



I is



SELECTION RULE: 5

PAINTING RULES

$L_5 \Rightarrow$ DK. BLUE

$L_4 \cap \sim L_5 \Rightarrow$ WHITE

$L_3 \cap \sim (L_4 \cup L_5) \Rightarrow$ RED

$L_2 \cap \sim (L_3 \cup L_4 \cup L_5) \Rightarrow$ DK. GREEN

$L_1 \cap \sim (L_2 \cup L_3 \cup L_4 \cup L_5) \Rightarrow$ YELLOW

$\sim (L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5) \Rightarrow$ LT. BLUE

LIMITING SHAPE

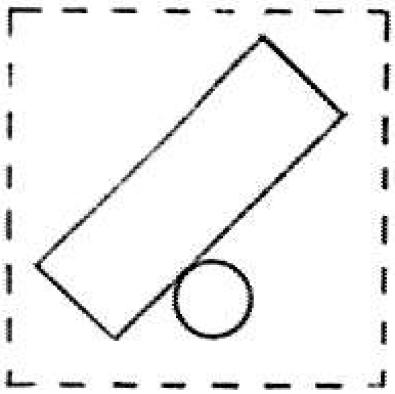


Figure 44. Generative specification for Aleatory.

3.2 Aesthetics

3.2.1 Original motivation

The investigation of aesthetics to be described was motivated by a curiosity about criteria used to evaluate paintings defined by generative specifications. Whenever a generative specification was created and the resulting painting generated, it inevitably would be evaluated. Operationally, the evaluation either could take the form of explicit statements of approval or disapproval of the generative specification and painting or else could be implicit in the final status of the generative specification and painting. Namely, some of the generative specifications and resulting (sketches of) paintings would end up in the wastebasket, some would be saved, some actually would be painted. With the implementation of the computer program just described, the process of creating generative specifications was facilitated and the process of generating and displaying the resulting paintings was automated. The human evaluation of the paintings remained. Some of the generative specifications were forgotten, some were saved on the disk. Some of the displayed paintings were photographed. The process of evaluating the paintings was intriguing, to say the least. What criteria were being used to evaluate the generative specifications and paintings? Could these evaluative criteria be specified? Could a computer program be written to do the evaluation? On investigating these questions, it soon became apparent that they could not be answered in isolation. Some very fundamental issues about aesthetics had to be examined first.

3.2.2 General observations on aesthetics

The first observation that must be made is the great variety of aesthetic viewpoints that exist, viewpoints which are frequently mutually inconsistent. Different people may like different works of art. The same work of art may be liked by someone and disliked by someone else. One person may consider a work of art to be a masterpiece while another considers it to be at best mediocre. To some extent, there may be a shared aesthetic viepoint, but this viewpoint may change over time or from culture to culture or from art form to art form. Given the multiplicity of aesthetic viewpoints, it would be foolish to attempt to define any absolute and universal aesthetic viewpoint. What does seem feasible is to characterize the logical properties and components of coherent, consistent aesthetic viewpoints. The variety of aesthetic viewpoints does not preclude the possibility of precisely stating aesthetic viewpoints. Crystallizing the notion of aesthetic viewpoint seems key.

A second observation is that the evaluation of a work of art is logically dependent on the interpretation of the work of art. For example, a painting of the type shown in the preceding sections may be interpreted by a viewer in many different ways, e.g. in terms of the colors of the painting or the structure of the shapes used in the painting or some association or emotion evoked by the painting. The painting may be interpreted as a political statement or as an abstraction of a face. How the painting is interpreted radically effects how it is evaluated. To specify an aesthetic viewpoint it is necessary to specify both the interpretative conventions and the evaluative criteria of the viewpoint.

Again, there are a great variety of interpretative conventions that are used. Most of the literature in art theory and criticism seems to be about interpretative conventions, e.g. discussions about specific interpretative conventions or descriptions of how particular works of art can be interpreted. An intriguing distinction between two types of interpretative conventions keeps resurfacing in the literature. This is the distinction between interpretative conventions which deal with the "internal coherence" of works of art and interpretative conventions which deal with the "external evocations" of works of art. The terms "internal coherence" and "external evocations" are original. Related, but somewhat discredited, terms are "form" and "content". Beardsley [1958] distinguishes between "critical description" and "critical interpretation". Frye [1957] contrasts "inward or centripetal" and "outward or centrifugal" interpretation. The distinctions made in the literature are not identical, but the idea is common. There is a difference between interpretation concerned with the internal structure of works of art and interpretation concerned with their external meaning. Internal coherence describes interpretative conventions dealing with composition, form, structure, organization, internal logic. Focillon [1948] and Arnheim [1954] provide interesting discussions of internal coherence in the visual arts. Interpretation concerned with internal coherence deals with intra-object relationships, perhaps with the relationships of parts of the object to each other and to the object as a whole. Interpretation of the paintings of Section 3.1 in terms of their shape grammars and generative specifications would deal with internal coherence. External evocation denotes interpretative conventions that deal with what a work of art means, what

objects, scenes or events are represented, what associations are made, what emotions are aroused or expressed. Gombrich and Goodman provide interesting discussions of external evocation in the visual arts in terms of expression [Gombrich 1963] [Goodman 1968] and representation [Gombrich 1960] [Goodman 1968]. Interpretation concerned with external evocations deals with extra-object relationships, with the relationships of the object and the outside world. Of course, all aesthetic viewpoints do not have interpretative conventions that are purely of one type or the other. Many authors (e.g. [Arnheim 1954]) stress the importance of using interpretative conventions that combine both internal coherence and external evocation. But the distinction is common and useful.

Returning to evaluation, it has been observed that the evaluation of a work of art is not absolute and universally agreed upon but relative to a specific aesthetic viewpoint and in particular that evaluation depends on interpretation. A striking feature of the literature on aesthetic evaluative criteria is the ubiquity of the terms "unity" and "variety" and related terms such as "order" and "complexity". The notion of aesthetic value being related to "unity in variety" seems to date back to the Greeks. A modern treatment of this notion is given in [Beardsley 1958]. An important work on this subject is the book Aesthetic Measure by the mathematician G. D. Birkhoff [1932]. Birkhoff defines the evaluative criteria $M = O / C$, where M is aesthetic measure, O is order and C is complexity. Birkhoff applies this measure to several classes of objects, e.g. Greek vases, polygons, music, poetry, by defining formulas for measuring the order and complexity of elements of each of the classes. For example, for polygons order is defined as

O = V + E + R + HV - F, where V is a measure of vertical symmetry, E is a measure of "equilibrium", R is a measure of rotational symmetry, HV is a measure of the relation of the polygon to a horizontal-vertical network, and F is a general negative factor which takes into account e.g., angles too near 0 degrees or 180 degrees and too small distances between vertices. Complexity, C, is defined as the number of indefinitely extended straight lines required to contain all the sides of the polygon. The ratio of the amount of order measured to the amount of complexity measured for each object is the aesthetic value assigned to the object relative to the class. This measure is then used to aesthetically order the objects in the particular class. The aesthetic measure defined for polygons was applied to ninety different polygons. Birkhoff's work can be criticized in terms of the lack of attention paid to the notion of interpretation, the arbitrariness of the way order and complexity were measured in the different classes of objects, and the belief that a single formula would be universally applicable, but it certainly is praiseworthy for its exactness, a quality notably missing in the rest of the literature. It is also notable for being universally ignored in the aesthetics and art literature.

To reiterate: There is a multiplicity of aesthetic viewpoints. It is infeasible to define a single, absolute, universal aesthetic viewpoint, but the logical characteristics of aesthetic viewpoints can be explored. Perhaps an example of an aesthetic viewpoint can be specified precisely. Works of art can be evaluated only relative to an aesthetic viewpoint. Evaluation depends on interpretation. An aesthetic viewpoint contains both interpretative conventions and evaluative

criteria. Two kinds of interpretative conventions have been distinguished, those concerned with the internal coherence of works of art and those concerned with their external evocations. Evaluation frequently has been discussed in terms of "unity" and "variety" or, similarly, "order" and "complexity". With few exceptions, the literature is remarkable for its general imprecision.

The final observation to be made is that aesthetic viewpoint is involved in the synthesis (design) of a work of art as well as the analysis of a work of art. Both the artist and the viewer/critic have an aesthetic viewpoint. In synthesis, an object which can be interpreted in such a way that it has a high aesthetic value from some aesthetic viewpoint is constructed. In analysis, a given object is interpreted in such a way that it has as high an aesthetic value as possible. In the following sections a method is developed that allows an aesthetic viewpoint to be specified in terms of its logical components and their interrelationships, independent of whether it is to be used for analysis or synthesis.

First, aesthetic system is defined. Aesthetic systems provide a logical framework in which the interpretative conventions and evaluative criteria of individual aesthetic viewpoints can be specified precisely. A preliminary discussion of aesthetic systems is given in [Gips and Stiny 1973]. In latter sections, a specific aesthetic system for paintings definable by generative specifications and its implementation on the computer is described. Aesthetic systems are related to Kolmogorov's formulation of information theory. Their applicability in the sciences is touched upon. Finally, the use of aesthetic systems in the analysis and synthesis of works of art is investigated.

3.2.3 Aesthetic systems

The definition of aesthetic system provides a logical framework in which divers aesthetic viewpoints can be formalized. The definition is an hypothesis about the underlying structure of possible aesthetic viewpoints. The assumption is that while the contents of specific aesthetic viewpoints may vary widely and be mutually inconsistent, the abstract organization of formalizations of these viewpoints may be the same. Each aesthetic system is a formalization of a particular aesthetic viewpoint. Given the multiplicity of aesthetic viewpoints and the inconsistency between different viewpoints, a universal aesthetic system is neither expected nor desired. The aesthetic system to be defined for paintings having generative specifications is a formalization of just one of many possible aesthetic viewpoints for interpreting and evaluating these paintings. While the definition of aesthetic system was developed as a framework in which this specific aesthetic viewpoint could be formalized, it is hoped that the definition provides a logical framework adequate for the formalization of many different aesthetic viewpoints for a wide variety of art forms.

It must be emphasized at the outset that the definition of aesthetic systems provides only a framework for formalizing different aesthetic viewpoints. There is no intention to gloss over the extremely difficult problems involved in specifying individual aesthetic viewpoints in terms of aesthetic systems. There is no magic here. The difficult part of specifying aesthetic viewpoints remains after the general definition of aesthetic systems is given. How to specify the components of

an aesthetic system so that the aesthetic system corresponds to a particular aesthetic viewpoint held by a person is a very open question that is likely to remain open for a long time.

An aesthetic system consists of four parts: (1) a set of interpretations I_A defined by an algorithm A, (2) a reference decision algorithm R which determines if an interpretation in I_A refers to a given object, (3) an evaluation function E which assigns values to interpretations in I_A, and (4) an order O which orders the values assigned to interpretations by the evaluation function. Formally, an aesthetic system is given by the 4-tuple $\langle I_A,R,E,O\rangle$.

For each aesthetic system, the set of interpretations I_A is defined abstractly by a fixed, deterministic algorithm A. I_A contains all possible input-output pairs $\langle\alpha,\beta\rangle$ for the algorithm A, where both α and β are finite, non-empty sequences of symbols from possibly different finite alphabets. The pair $\langle\alpha,\beta\rangle$ is an interpretation in the set I_A if and only if when given input sequence α, A terminates with output sequence β. Membership of an interpretation in I_A is independent of actual objects, depending only on the sequences α and β and the algorithm A. I_A is a potentially infinite set of finite interpretations. Intuitively, I_A contains all interpretations that possibly could refer to objects from the aesthetic viewpoint specified by the aesthetic system.

An interpretation has two parts, α and β. Generally, one is a description of an object and the other is a specification of how that description is understood. Two basic types of interpretation are distinguished which correspond to the traditional division between external evocation and internal coherence (see

previous section). In an interpretation examining the external evocation of an object, the description of the object is paired with the response evoked by the object. In an interpretation examining the internal coherence of an object, a specification of the underlying structure of the object is paired with the description of the object. To make this more precise requires a discussion of the reference decision algorithm.

The reference decision algorithm R when presented with an interpretation in I_A and any real-world object decides whether the interpretation refers to the object. The notion of "object" is used here in its widest possible sense to include, for example, musical or theatrical performances as well as paintings or novels. Or elephants for that matter. The general form of the reference decision algorithm is shown in Figure 45. R contains a sensory input transducer, S, which provides an interface with the object. The object is presented to the sensory input transducer (which for painting might contain a high resolution color television camera) while an interpretation $\langle\alpha,\beta\rangle$ in I_A is given as input. The output of R is True if and only if the interpretation refers to the object. In general, there are many interpretations in I_A which do not refer to actual objects. It is possible that for a given aesthetic system, every object would have some interpretation, however minimal, that would refer to it. Alternatively, for a given aesthetic system only a very limited class of objects (e.g., paintings of a certain restricted type) might have interpretations which refer to them. A discussion of reference in the context of aesthetics is given in [Goodman 1968].

The reference decision algorithm allows for the precise definition of "work of

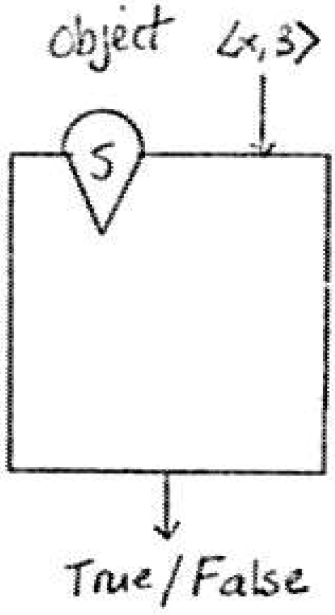


Figure 45. General structure of reference decision algorithm.

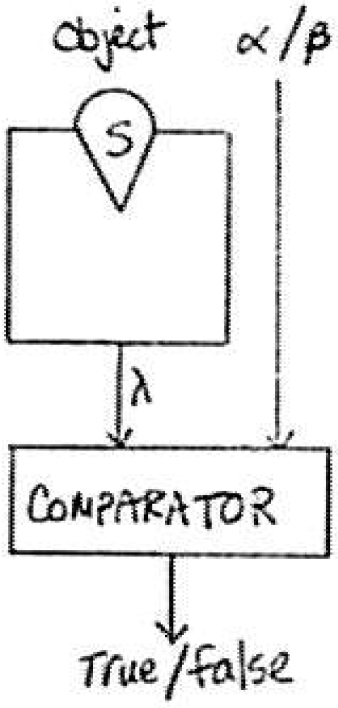


Figure 46. Special reference decision algorithm schema.

art": for a given aesthetic system, an object is a work of art if and only if there is an interpretation in the set of interpretations which refers to it.

Additional structure for R can be specified. In particular, the use of the reference decision algorithm schema of Figure 46 results in interpretations which deal with either the external evocations or internal coherence of objects to which they refer. For an interpretation $\langle\alpha,\beta\rangle$, either α or β, but not both, is used as input for the determination of reference in this schema. The first part of the schema, shows a sensory input transducer, S, linked to an algorithm which produces a finite and discrete canonical description, λ, of the presented object. For example, in music, drama, literature, or architecture λ could resemble the score, script, text, or plan. λ is the complete description of the object in the sense that only those attributes identified by λ are considered in interpretations. Different objects producing identical λ are indistinguishable for interpretation in a given aesthetic system of this type. This can allow a single interpretation to refer to multiple reproductions of a painting, copies of a novel, or performances of a concerto. Or, in a given aesthetic system, it can allow a single interpretation to refer to all Bartok piano pieces or all Jackson Pollock paintings. The second part is a comparator which has as output True if λ is identical to the input component of the interpretation and False otherwise. This reference decision algorithm schema gives an operational definition of the canonical description, λ, of an object. The two cases of this schema, i.e., where $\alpha=\lambda$ and where $\beta=\lambda$, are paradigmatic for the two basic types of interpretation distinguished above.

Aesthetic systems in which α in interpretations $\langle\alpha,\beta\rangle$ is the description of an

object, e.g., $\alpha=\lambda$ in the reference decision algorithm schema of Figure 46, may be considered to deal with the external evocations of objects referenced by interpretations in I_A. For interpretations in these systems, β is a specification of the response evoked by this description. Given α, the algorithm A determines explicitly the entire content of this response. The description of an object is the input to the algorithm; the response to that description is the output of the algorithm. For example, β may be a symbolic encoding of the associations or emotions evoked by the description of the object. The algorithm A embodies the interpretative conventions of a specific aesthetic viewpoint which determine what associations or emotions are attached to the description of an object. Discussions of an object in terms of content, representation, expression, etc. can be based on interpretations of this type. Some possible aesthetic systems which deal with the external evocations of paintings definable by generative specifications are discussed in Section 3.2.9. Again, it must be emphasized that the definition of aesthetic systems provides only a framework for specifying aesthetic viewpoints. Constructing an aesthetic system that corresponds to a particular aesthetic viewpoint held by a person (especially a viewpoint of the type just discussed) can be extremely difficult.

Aesthetic systems in which β in interpretations $\langle\alpha,\beta\rangle$ is the description of an object, e.g., $\beta=\lambda$ in the reference decision algorithm schema of Figure 46, may be considered to deal with the internal coherence of objects referenced by interpretations in I_A. For interpretations in these systems, α is a specification of β, the description of the object, in terms of its syntactic or semantic structure. α is a sequence of symbols which when processed by the algorithm A produces

exactly β, the description of the object. The algorithm A determines implicitly the nature of acceptable underlying structures for the description of the object. For example, β could be specified by α consisting of a symbolic encoding of rules of construction or principles of organization based on the occurrence of patterns, motifs, or themes. The algorithm A embodies the interpretative conventions of a specific aesthetic viewpoint which determine how the description of an object can be constructed from α. Discussions of an object in terms of form, composition, etc. can be based on interpretations of this type. The aesthetic system for paintings having generative specifications presented in section 3.2.4 is an example of this kind of aesthetic system.

An aesthetic system which deals with internal coherence and an aesthetic system which deals with external evocations sometimes can be composed to form a single aesthetic system. A preliminary discussion of this construction is given in [Gips and Stiny 1973]; a detailed discussion is given in [Stiny, in preparation]. This new aesthetic system deals with both the internal coherence and external evocations of objects referenced by interpretations. In this system, α in an interpretation which refers to an object is a symbolic encoding of the rule of construction or organization of the object; β gives the evocations of the object. The new algorithm A is the composition of the algorithms of the two original aesthetic systems and produces the description of the object internally. In practice, most aesthetic viewpoints seem to be of this type, dealing with both internal coherence and external evocations.

A set of interpretations I_A and a reference decision algorithm R are a

formalization of particular interpretative conventions. I_A defines the potential scope of thse conventions; R determines their empirical extent. Any interpretative conventions are allowable if A and R can be constructed to conform to them.

The evaluation function E is defined on the set I_A and assigns an aesthetic value to each interpretation in I_A. There are many possibilities for evaluation functions. An evaluation function for interpretations $\langle\alpha,\beta\rangle$ may be defined in terms of just α, just β, or both. The function may consider the content of an interpretation, e.g., the occurrence of specific symbols in α or β, or the general characteristics of an interpretation, e.g., the lengths of α or β. O is an order defined in the range of E and may be partial or total. The evaluation function together with the order ranks elements of I_A. Note that direct aesthetic comparisons can be made only within a given aesthetic system.

An interesting evaluation function for aesthetic systems is given by

$$E_Z(\langle\alpha,\beta\rangle) = L(\beta)/L(\alpha)$$

where $L(\alpha)$ is the length of α and $L(\beta)$ is the length of β. The total order O_Z naturally associated with E_Z would rank two interpretations such that the interpretation assigned the higher value is aesthetically superior.

Because the evaluation function E_Z is defined in terms of the lengths of α and β occurring in interpretations, E_Z and O_Z can be combined with any set of interpretations I_A and reference decision algorithm R to form an aesthetic system. For a given I_A, the evaluation function E_Z assigns high aesthetic values to interpretations $\langle\alpha,\beta\rangle$ in which α is an economical specification of β with respect to A, i.e. in which α is much shorter than β.

For aesthetic systems which deal with external evocation, in an interpretation $\langle\alpha,\beta\rangle$ which refers to an object, α is a description of the object and β is an encoding of what is evoked by this description (see Figure 46). In this type of aesthetic system, an interpretation $\langle\alpha,\beta\rangle$ is assigned a high aesthetic value by E_Z if the symbolic encoding of the evocations, β, of the object are very long relative to the description, α, of the object. Here the description may have multiple, lengthy evocations. The input to the algorithm (i.e. the description of the object) is much shorter than the output. An interpretation in this type of aesthetic system is assigned low aesthetic value by E_Z if the description has minimal evocations.

For aesthetic systems which deal with internal coherence, in an interpretation $\langle\alpha,\beta\rangle$ which refers to an object, α is a specification of β, the description of the object. In this type of aesthetic system, an interpretation $\langle\alpha,\beta\rangle$ is assigned high aesthetic value by E_Z if the rules of organization or construction, α, is very short relative to the description, β, of the object. Here, the description of the object has a brief encoding in terms of the algorithm A. The input to the algorithm is much shorter than the output (i.e. the description of the object). The description of the object has high internal coherence relative to A. An interpretation in this type of aesthetic system is assigned low aesthetic value if the rules of organization, α, are lengthy compared to the description, β.

The use of the evaluation function E_Z in aesthetic systems concerned with internal coherence can be considered a precise formulation of the traditional notions of "unity" and "variety" in aesthetic evaluation (see previous section). In this context, the "unity" of the object is measured by $L(\alpha)$ as it indicates the

brevity of the rules of construction or principles of organization required to obtain β, the description of the object, using A. The smaller the length of α, the greater the "unity". $L(\beta)$, the length of the description of the object, provides a measure of the variety of the object. The greater the length of β, the greater the "variety". Using this characterization, if an object has "unity" and "variety" relative to an aesthetic system, then an interpretation which refers to it contains a short α and a long β and therefore is assigned high aesthetic value by E_Z.

There are other possible evaluation functions that are generally applicable and are of interest. For example, it has been suggested that the amount of computation required for the algorithm A to output β given α as input should be taken into account in evaluating the interpretation $<\alpha,\beta>$. The idea is that, in general, interpretations in which relatively brief computation is required to compute β given α are aesthetically preferred. A first approximation of an evaluation function that combines this notion with the evaluation function E_Z might be $E_{TA}(<\alpha,\beta>) = E_Z(<\alpha,\beta>)/t_A(\alpha,\beta) = L(\alpha)/(L(\beta)*t_A(\alpha,\beta))$ where $t_A(\alpha,\beta)$ is some measure of the time required for the algorithm A to compute β given α. Notice that this evaluation function is dependent on the exact nature of the algorithm A, whereas E_Z is dependent only on the characteristics of the input-output pairs of the algorithm. Left unspecified are exactly how t_A is to be computed and the relative weights to be assigned E_Z and t_A. The idea of including a measure of the computation time in the evaluation function will not be pursued further here, nor will other possible evaluation functions be explored. The evaluation function E_Z will be used in most of what follows.

3.2.4 Aesthetic systems and information theory

During the past twenty-five years there have been numerous attempts to apply results in information theory (and related earlier results in thermodynamics) to aesthetics and art theory, e.g. [Arnheim 1971] and [Meyer 1959]. This work has been hampered by two serious problems. First, the notions of aesthetics have been extremely imprecise. Second, the work has attempted to apply a statistically-oriented formulation of information theory to individual works of art. These problems can be remedied by using the precise notions of aesthetic systems just described and the algorithm-oriented formulation of information theory developed by Kolmogorov [1968], Solomonoff [1964], Chaitin [1970], and others. This new formulation provides for the computation of the information-theoretic entropy of an individual sequence of symbols in terms of the length of the shortest algorithmic specification of the sequence instead of the likelihood of occurrence of the sequence among all possible sequences. The entropy of an individual sequence can be calculated in terms of its own structural properties independent of the content and statistical properties of the full set of possible sequences.

Assume that α and β are defined over binary alphabets and that A is a universal computing algorithm. The entropy of the sequence β with respect to the algorithm A, $H_A(\beta)$, is defined by Kolmogorov [1968] to be the length of the shortest sequence α which when used as input to A produces output β. When this sequence α is shorter than β (i.e. $L(\alpha) = H_A(\beta) < L(\beta)$), β must have some structural regularity or periodicity. When this sequence α is the same length or

longer than a (long) sequence β (i.e. $L(\alpha) = H_A(\beta) \geq L(\beta)$), β has no strutural regularity or periodicity and may be considered random [Kolmogorov 1968] [Martin-Lof 1965]. Note that no sequence β can have entropy much greater than its length as α can always be the instruction "write out β". It is easy to show that there are relatively few sequences with low entropy and that almost all sequences have nearly maximal entropy [Martin-Lof 1965].

The application of these results to aesthetic systems is straightforward and provides insight into the evaluation function E_Z. For an interpretation $\langle\alpha,\beta\rangle$ in a set of interpretations I_A in an aesthetic system using the evaluation function E_Z, if α is the shortest input sequence which produces output β when processed by A, then E_Z computes the ratio of the length of β to its entropy (i.e. $L(\alpha) = H_A(\beta)$ and $E_Z(\langle\alpha,\beta\rangle) = L(\beta)/L(\alpha) = L(\beta)/H_A(\beta)$). The aesthetic value assigned this interpretation is approximately the entropy of a random sequence of length $L(\beta)$ divided by the actual entropy of β. This expression can be considered the reciprocal of the relative entropy [Shannon and Weaver 1948] of β. Any interpretation in I_A that is assigned aesthetic value greater than 1 by E_Z must contain β that is not random. If the aesthetic value assigned an interpretation is greater than 1, the entropy of β must be shorter than its length. Any interpretation in I_A that contains β which is random is assigned low aesthetic value (≤ 1) by E_Z. These results are especially interesting for aesthetic systems in which β is the description of an object. In this case, all interpretations with object descriptions that are random sequences are assigned low aesthetic values by E_Z. From the final sentence of the preceding paragraph it follows that a set of

interpretations I_A is, in general, very sparse in interpretations to which E_Z assigns high aesthetic value (i.e. for which $E_Z(<\alpha,\beta>) >> 1$).

3.2.5 An aesthetic system for paintings definable by generative specifications

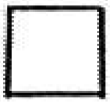
An aesthetic system which contains interpretations which refer to the paintings specifiable using the program for generative specifications described in Section 3.1.2 has been developed and implemented in part on the computer.

In an interpretation $<\alpha,\beta>$ in this aesthetic system, α is a generative specification that can be defined using the program. β is a straightforward and exhaustive description of the resulting painting made in terms of the shapes and colors of the different painted areas of the painting. β takes into account only the most obvious shape and color redundancies that occur in paintings of this type. The key to β is that it can be constructed from just an image of the painting as well as from a generative specification.

β in an interpretation in this aesthetic system consists of three tables - a shape table, a color table, and an occurrence table. The format of these tables is shown in Figure 47.

The shape table specifies the different shapes of the painted areas in the painting. There is one entry in the shape table for each geometrically non-similar shape. For example, if any of the areas of the paintings are squares, exactly one entry in the shape table would be a square.

SHAPE TABLE

i_s	Shape
1	
2	
.	.
.	.
.	.

OCCURRENCE TABLE

i_s	i_c	x	y	θ	s	m
i_{s_1}	i_{c_1}	x_1	y_1	θ_1	s_1	m_1
i_{s_2}	i_{c_2}	x_2	y_2	θ_2	s_2	m_2
			.			
			.			
			.			

COLOR TABLE

i_c	Color
1	red
2	blue
.	.
.	.
.	.

Figure 47. Format of tables of β.

The color table specifies the different colors of the areas in the painting. There is one entry in the color table for each color. For example, if some of the areas in the painting are painted a certain shade of blue, exactly one entry in the color table would be that color. Entries in the color table for a painting normally are identical to the colors specified in the right side of the painting rules of the generative specification of the painting.

Each entry in the occurrence table corresponds uniquely to a distinct colored area occurring in the painting. Each entry has seven parts: i_s is the index of a shape entry occurring in the shape table and specifies the shape of the area; i_c is the index of a color entry occurring in the color table and specifies the color of the area; x, y, θ, s, and m are transformations which map the shape indexed by i_s from the shape table coordinate system to the painting coordinate system, where x and y determine translation, θ determines rotation, s determines scale, and m determines if the mirror image of the shape is used. For example, assume a blue square occurs in the top right corner of the painting. This means that one of the entries of the shape table is a square and one of the entries in the color table specifies the shade of blue. In the entry in the occurrence table that specifies this particular colored shape, i_s is the index number of the entry for square in the shape table, i_c is the index number for the entry for blue in the color table, and x, y, θ, s, and m specify the exact location of the square in the painting.

An example of the shape, color, and occurrence tables for an actual painting is given in Section 3.2.7.

Interpretations in this aesthetic system deal with the internal coherence of

paintings. α, a generative specification, is an indication of the underlying structure of the painting. β is a description of the painting. The algorithm A, given an α as input, produces the corresponding β as output.

The reference decision algorithm R in this aesthetic system can be constructed to correspond with the β form of the reference decision algorithm schema of Figure 46. The sensory input transducer S would be a color television camera. The algorithm linked to S would contain a color oriented edge-following or region-growing routine. The description λ constructed by this algorithm for the object would have the table format of β. The β of any interpretation which refers to an object would be identical to the description λ constructed for the object in the reference decision algorithm. The reference decision algorithm has not been implemented on the computer because it is not necessary for the specific task which the program performs.

In this aesthetic system, the evaluation function $E_Z(<\alpha,\beta>) = L(\alpha)/L(\beta)$ is used as is the associated order O_Z. In the calculation of aesthetic value using E_Z, the lengths of α and β are defined to be the number of words of memory used to represent them in the computer implementation.

3.2.6 Computer implementation

Parts of the aesthetic system just described have been implemented on the computer. The aesthetics program has been incorporated into the program for generative specifications described in Section 3.1.2. The net result is that an aesthetic value can be calculated by the program for the interpretations of many paintings definable using the program. Looking at the whole program in aesthetic system terms, the program is given α (a generative specification) and can be asked to (1) use the algorithm A to construct the resulting β (tables), (2) evaluate this interpretation $\langle\alpha,\beta\rangle$ using E_Z, and (3) construct an object (display the painting) to which the interpretation refers. Thus the algorithm A and the evaluation function E_Z have been implemented. The reference decision algorithm has not been implemented because it is assumed that the program works, namely that the constructed interpretation really does refer to the displayed painting. The order O_Z has not been implemented because the program works with only one interpretation and one painting at a time. The user must make any comparisons of aesthetic value.

The aesthetic program can be called from the monitor level described in Section 3.1.2.1, i.e. after a generative specification has been defined and a shape generated using the shape grammar and selection rule. Given a generative specification and specified shape, the aesthetics program (1) constructs the shape, color, and occurrence tables (i.e., β) for the painting and (2) calculates the ratio of the length (number of words of memory used in the computer representation) of

the length of β (the tables) to the length of α (the generative specification) to obtain the aesthetic value assigned by E_7.

To construct the shape and occurrence tables, the aesthetics program must first derive the outline of the different colored areas of the final painting. The shape generated by the shape grammar consists of multiple instances of the terminal shape. To determine the outline of the areas in the final painting, the effect of the painting rules must be taken into account. For example, terminals at the same level may overlap and have to be merged. Parts of shapes at the lower levels may be hidden by shapes of a higher level. This part of the program makes use of variations of the algorithms reported in [Eastman and Yessios 1972] for performing Boolean operations on shapes. The implementation is quite involved and uses a temporary but lengthy data structure of nodes and pointers. Because of this, there is a limit on the number of lines and shapes in paintings for which the aesthetic program can be run. Also, the current program does not work properly for some paintings containing shapes with complicated configurations of holes. It is expected that these restrictions will be alleviated in future versions of the program.

The internal representation of the shape, color, and occurrence tables is important for the calculation of aesthetic value in this aesthetic system. The evaluation function E_7 is defined as the ratio of the length of β to the length of α, where the lengths of α and β are the number of words of memory used to represent them in the program. The number of words of memory used to represent α, a generative specification, in the program is described in detail in

Section 3.1.2.6. The number of words used to represent β is the sum of the number of words used to represent its three components - the shape table, the color table, and the occurrence table.

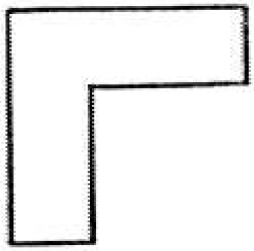
Each entry in the shape table is a closed, rectilinear shape. The representation of each entry is constructed as follows:

(1) Find the shortest edge of the shape and call its length 100.

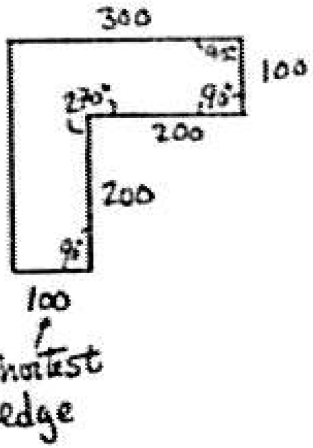
(2) Trace around the shape counter-clockwise beginning at the vertex after the shortest edge. At each vertex record the angle between the edges. At each edge record the length of the edge assuming the shortest edge is 100. Stop the trace after reaching the next to last edge. The trace can be stopped here because the shape is known to be closed and rectilinear so the final angles and edge length are redundant.

In step (1), if there is more than one edge of the shortest length, the tie is broken by beginning a counter-clockwise trace from each of the short edges and successively comparing each angle and edge length encountered. The short edge that is followed by the smallest angle (and if that is a tie, the smallest edge, etc.) is selected. If an entire trace is made and the tie is not broken, which short edge is chosen does not matter: the shape is symmetric. Thus, each closed, rectilinear shape is represented as a list of (angle, distance) pairs. The representation must also include one word which gives the number of pairs listed. An example of the representation of a shape is given in Figure 48. For a shape with a hole, two identical edges are added initially between a vertex on the inner boundary and a vertex on the outer boundary so that the trace around the shape is continuous.

SHAPE:



ANGLES AND LENGTHS ENCOUNTERED:



REPRESENTATION IN SHAPE TABLE:

4 (90, 200) (270, 200) (90, 100) (90, 300)

Figure 48. Representation of a shape in a shape table.

The number of words of memory used to represent each entry in the shape table in this format is 1+2(V-2) = 2V-3, where V is the number of edges of the shape. This representation was chosen because it is invariant under the transformations of translation, rotation, and scale and thus facilitates comparison of two shapes of arbitrary position, orientation, and size. The number of words used to represent the shape table is the sum of the number of words used for each entry plus one word which gives the total number of entries in the table.

Each entry in the color table is a color and is represented by three words, one each for the red, blue, and green intensity components of the color. These three components correspond to the way color is specified for the program that displays generated paintings on the color television (see Section 3.1.2.5). The number of words used to represent the color table is three times the number of entries (number of colors in the painting) plus one word which gives the total number of entries in the table.

Each entry in the occurrence table has seven parts: i_s the index of the shape in the shape table, i_c the index of the color in the color table, and x, y, θ, s, and m the parameters which specify the location of the shape on the canvas. Each entry is represented by seven words, one for each part. The number of words used to represent the occurrence table is seven times the number of entries (the number of different painted areas in the painting) plus the usual one word which gives the total number of entries in the table.

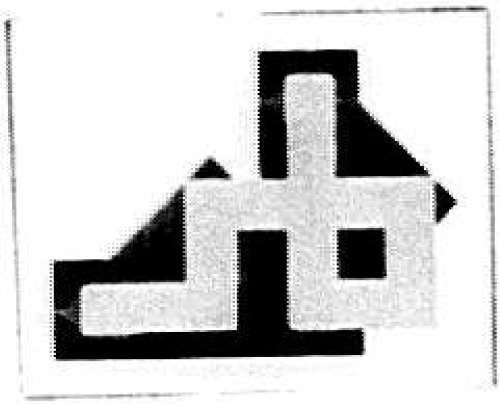
When the aesthetics program is called from the monitor, the program displays the outline of the areas of the final painting, outputs the lengths of α and β and the value assigned by E_Z, and then returns to the monitor level from whence it came.

3.2.7 Example: Anamorphism I - VI

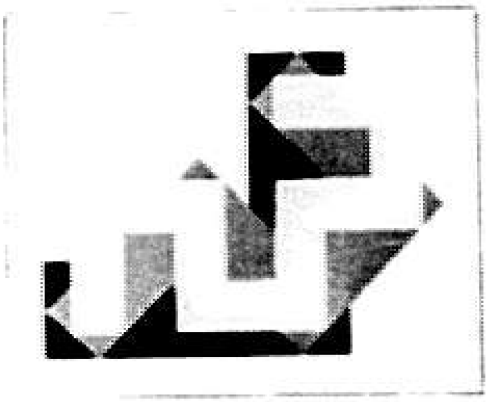
Six paintings, Anamorphism I - VI, are shown in Figure 49. In this section, interpretations are given for these paintings and these interpretations are ranked. This example was done using the program. Specifically, generative specifications for these paintings were defined using the program and the shapes generated. The aesthetics program then constructed the associated shape, color, and occurrence tables and calculated the aesthetic values assigned by E_Z. These values can be ordered using O_Z. The pictures of the paintings in Figure 49 are photographs of the computer display of the paintings.

The generative specification of Anamorphism I is shown in Figure 50. The generative specifications for Anamorphism II - VI can be obtained by substituting the rules shown in Figure 51 for the first rule in the shape grammar in the generative specification for Anamorphism I. Additionally, the limiting shapes of Anamorphism V and VI are slightly different than the others bcause of the differences in length and width of the generated shapes in these paintings. The generative specifications of these six paintings differ primarily in the location of the markers (circles) in the right side of the first rule in the shape grammar.

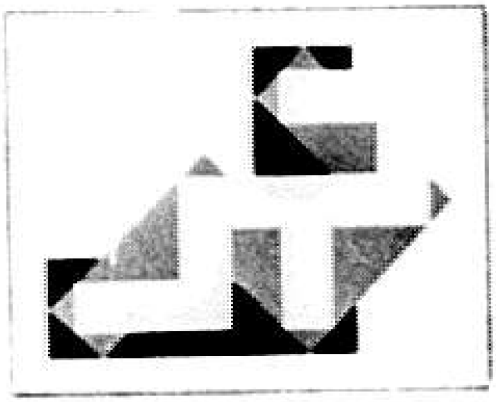
The lengths of the computer representations of these generative specifications are the same. In the computer representation of each of these generative specifications: The terminal is a rectangle; the number of vertices in the terminal shape (NVT) is 4. Two terminals are added by the right side of the rule; the number of instances of the terminal shape in the rule shape (NRS) is 2. The



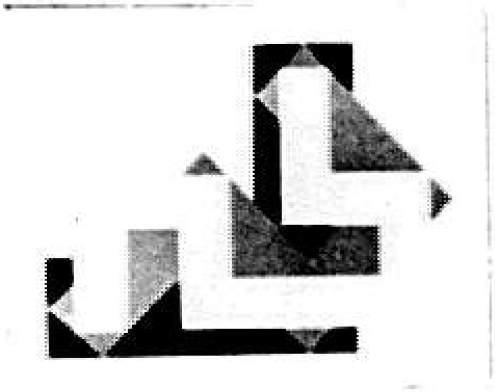
Anamorphism I



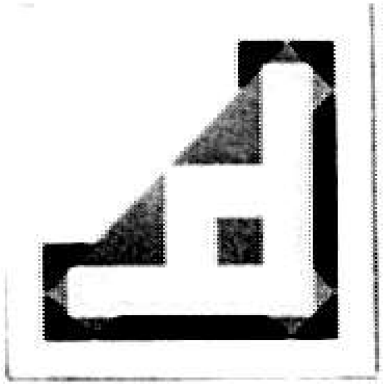
Anamorphism II



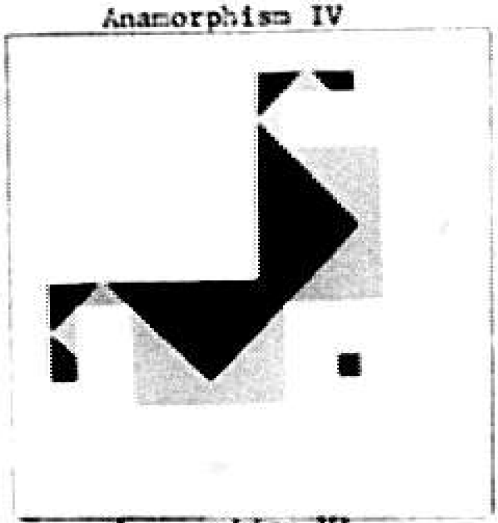
Anamorphism III



Anamorphism IV



Anamorphism V



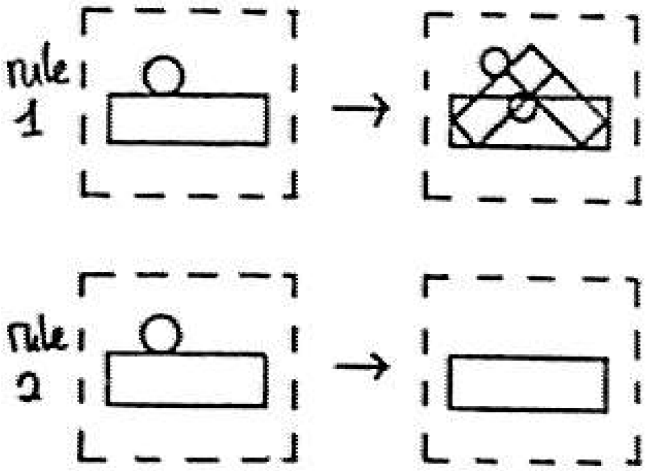
Anamorphism VI

Figure 49. Anamorphism I - VI. Colors are blue, red, green, light blue (darkest to lightest).

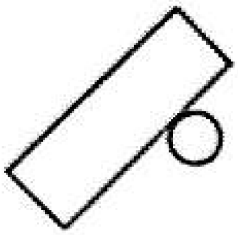
SHAPE GRAMMAR PSG18

$V_T = \{ \square \}$ $V_M = \{ \bigcirc \}$

R contains



I is



SELECTION RULE: 3

PAINTING RULES

$L_3 \Rightarrow$ GREEN

$L_2 \cap \sim L_3 \Rightarrow$ RED

$L_1 \cap \sim (L_2 \cup L_3) \Rightarrow$ BLUE

$\sim (L_1 \cup L_2 \cup L_3) \Rightarrow$ LT. BLUE

LIMITING SHAPE

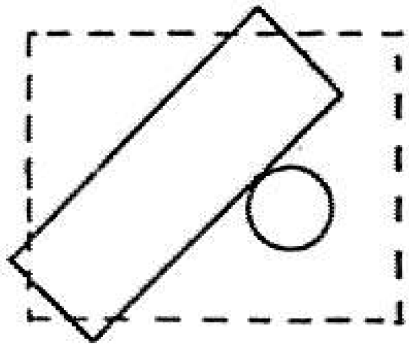
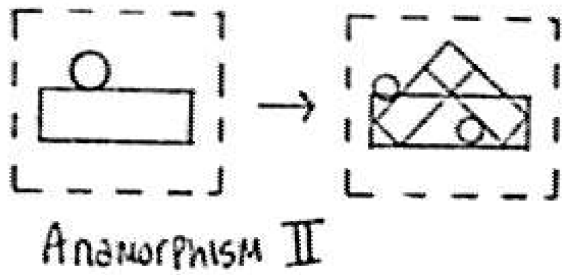
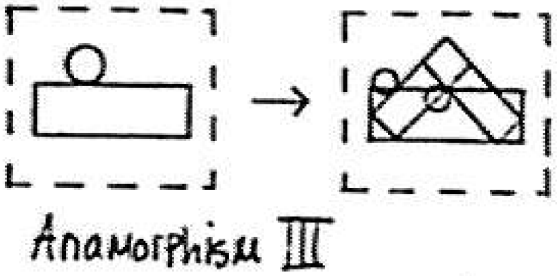
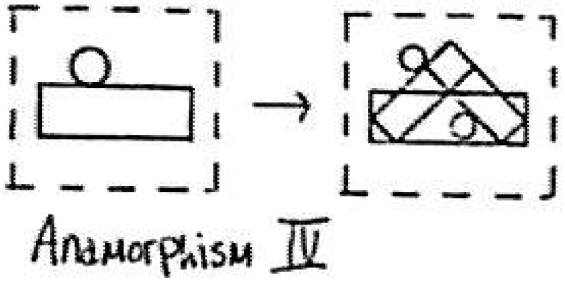
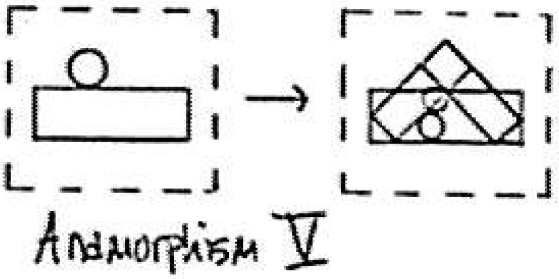


Figure 50a. Generative specification for Anamorphism I.









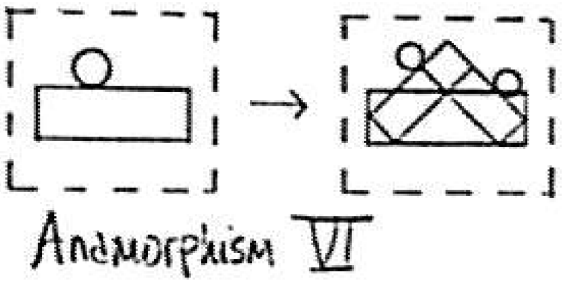


Figure 50b. First shape rules of generative specifications for Anamorphism II - VI.

initial shape contains one terminal; the number of instances of the terminal shape in the initial shape (NIS) is 1. The number of levels displayed in the painting (NLEV) is 3. In fact, the computer representations of the generative specifications differ only in the exact values of the θ (rotation) and m (mirror image) parameters of the terminal shapes in the rule shape. Using the formula developed in Section 3.1.2.6, the total number of words used to represent each of these generative specifications is 2*NVT + 5*NRS + 5*NIS + 3*NLEV + 9 = 2*4 + 5*2 + 5*1 + 3*3 + 9 = 41.

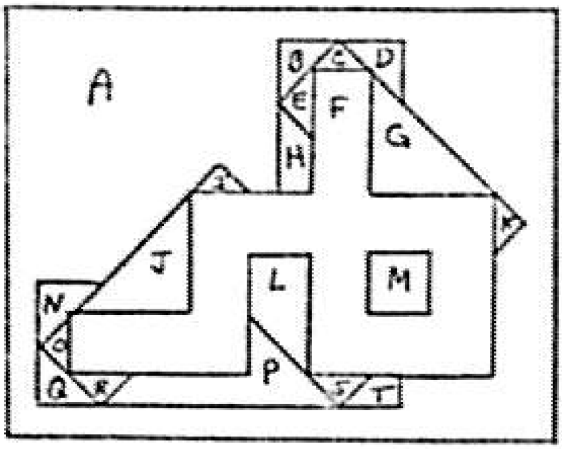
Since the lengths of the α in the interpretations that refer to Anamorphism I - VI are the same, the aesthetic value assigned by E_Z to each interpretation is directly proportional to the length of β in the interpretation.

The β (shape, color, and occurrence tables) constructed for Anamorphism I is shown in Figure 51. The shape table has seven entries, one for each of the shapes occurring in the painting. Calculation of the length of the shape table is shown in Figure 52. The color table has four entries, one for each color. The occurrence table has twenty entries, one for each of the distinct colored areas of the painting. The output of the aesthetics program for Anamorphism I is shown in Figure 53.

The number of entries and length (number of words used in the representation) of each of the tables of β for the six paintings are shown in Figure 54a. The aesthetic value assigned each of the interpretations for Anamorphism I - VI is shown in Figure 54b.

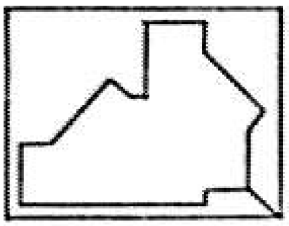
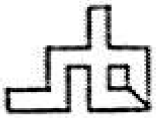
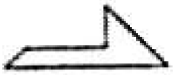
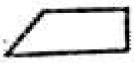
The ordering determined by O_Z of the aesthetic value assigned by E_Z to the given interpretations which refer to Anamorphism I - VI is, in order of decreasing aesthetic value, Anamorphism I, IV, II, III, VI, V.

Outline of Anamorphism I with painted areas lettered:



β constructed for Anamorphism I:

SHAPE TABLE

is	shape
1	
2	
3	
4	
5	
6	
7	

COLOR TABLE

ic	Color
1	GREEN
2	RED
3	BLUE
4	LT. BLUE

OCCURRENCE TABLE

is	ic	x	y	θ	s	M
1	4	Param's for area				A
7	3	"				B
7	2	"				C
7	3	"				D
7	2	"				E
2	1	"				F
7	2	"				G
6	3	"				H
7	2	"				I
7	2	"				J
7	2	"				K
4	2	"				L
5	2	"				M
7	3	"				N
7	2	"				O
3	3	"				P
7	3	"				Q
7	2	"				R
7	2	"				S
4	3	"				T

Figure 51. β for Anamorphism 1.

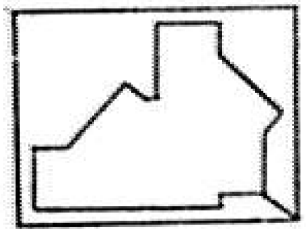
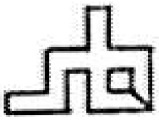
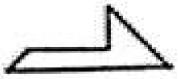
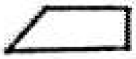
i_S	shapes	no. of edges V	length 2V-3
1		20	37
2		20	37
3		5	7
4		4	5
5		4	5
6		4	5
7		3	3
total			99
length of table (total+1)			100

Figure 52. Calculation of length of shape table of β for Anamorphism I.

α: SG	terminal shape	# lines = 4	length = 9
	rule shape	# shapes = 2	length = 11
	initial shape	# shapes = 1	length = 6
	selection rule	<1,3>	length = 2
	coloring rules	# levels = 3	length = 12
	limiting shape	border = 38	length = 1
			L(α) = 41
β:	shape table	# entries = 7	length = 100
	color table	# entries = 4	length = 13
	occurrence table	# entries = 20	length = 141
			L(β) = 254
			EZ(<α,β>) = 6.195

length of smallest side = 16
length of border = 235
required spatial resolution = 1/18

Figure 53. Output of aesthetics program for Anamorphism I.

Anamorphism	Shape Table		Color Table		Occur. Table		L(β)
	#entries	length	#entries	length	#entries	length	
I	7	100	4	13	20	141	254
II	6	95	4	13	20	141	249
III	5	82	4	13	20	141	236
IV	7	98	4	13	20	141	252
V	5	62	4	13	18	127	202
VI	6	95	4	13	16	113	221

Figure 54a. Summary of $\mathcal{L}$ for Anamorphism I - VI.

Anamorphism	L(α)	L(β)	EZ(<α,β>)
I	41	254	6.20
II	41	249	6.07
III	41	236	5.76
IV	41	252	6.15
V	41	202	4.93
VI	41	221	5.39

Figure 54b. Calculation of aesthetic values assigned the interpretations for Anamorphism I - VI in this aesthetic system.

3.2.8 Comments

The aesthetic system just described deals with the internal coherence of paintings having generative specifications. For interpretations $\langle\alpha,\beta\rangle$ in this aesthetic system, α is a specification of the underlying structure of the painting and β is the description of the painting. The algorithm A embodies the conventions by which β, the shape, color, and occurrence tables, can be constructed given α, a generative specification. The set of interpretations I_A contains all possible generative specifications definable using the program and their associated shape, color, and occurrence tables. β is easily obtainable from a painting, allowing for a straightforward construction of a reference decision algorithm R having the β form of the schema of Figure 46. The evaluation function, E_Z, used in this aesthetic system assigns high aesthetic value to interpretations having short generative specifications and long shape, color, and occurrence tables.

The aesthetic system is concerned primarily with shape. The aesthetic value assigned to an interpretation which refers to a painting in this aesthetic system reflects primarily the variety and number of shapes in the painting and the simplicity of the shape grammar used to generate them. It is interesting to note that there is an implicit bias in this aesthetic system against symmetric paintings. The bias results because asymmetric paintings tend to have a larger variety and more occurrences of shapes than symmetric paintings. Thus the shape and occurrence tables of asymmetric paintings tend to have more entries than the shape and occurrence tables of symmetric paintings and the β tend to be longer.

The aesthetic values assigned to the interpretations given for Anamorphism I - VI are an example of this phenomenom.

A (possibly unfortunate) characteristic of the aesthetic system is that aesthetic value is invariant under rotation of a painting. If a painting defined by a generative specification is turned, say, 90 degrees, this would be considered a new painting by the aesthetic system. This new painting would have a different β (shape, color, and occurrence tables) but the new β would be of the same length as the β for the original painting, differing only in the values of various parameters in the occurrence table. The α (generative specification) of the new painting can be obtained by roteting the initial shape and limiting shape of the original generative specificatien 90 degrees. This α for the new painting would have the same length as the original α. So the interpretations for the two paintings would be assigned the same aesthetic value in this aesthetic system.

In theory, just about any generative specification can be used to define a painting with an interpretation assigned an arbitrarily high aesthetic value in this aesthetic system by simply increasing the selection rule (and adding the appropriate painting rules). Increasing the selection rule increases the number of levels generated by the shape grammar. This generally has the effect of greatly increasing the lengths of the shape and occurrence tables of β but only minimally increasing the length of α, the generative specification. This does not seem desirable in the limit. At some point, I get saturated observing a painting containing a tremendous number of colored areas. For example, if the areas of the painting get very small (as they often do with a large selection rule), the areas seem to

blur together and have the effect of forming textures rather than distinct areas. Of course, in practice, the program breaks down with a large selection rule. The most complicated versions shown for Urform (Figure 39) and Star (Figure 41) use the full resources of the programs described in Section 3.1.2 in terms of storage space, computing time, and the spatial resolution of the displays. The aesthetics program currently works only for less complicated paintings as it needs more memory than the shape generation and display programs. A remedy for the theoretical unboundedness of aesthetic value with increasing selection rule might be to modify the evaluation function by taking into account computing time (as discussed briefly at the end of Section 3.2.3) or possibly by normalizing the aesthetic value in terms of the number of shapes or shape edges in the painting or the required spatial resolution. Or, the aesthetic system could be modified in theory to make it conform with what now occurs in practice. Namely, absolute bounds could be put on measures such as computation time, storage space for the tables of β, or the spatial resolution required to display the painting.

The reader may well disagree with the the aesthetic orderings made by the program. It should be re-emphasized that a universally agreed upon aesthetic system is neither expected nor desired. Alternative aesthetic systems for paintings definable by generative specifications are discussed in the next section. The aesthetic system used does embody a coherent, well-defined aesthetic viewpoint that seems like a reasonable first approximation of my own.

3.2.9 Alternative aesthetic systems for paintings definable by generative specifications

The aesthetic system that has been presented is just one of many possible aesthetic systems possible for paintings definable by generative specifications (and, of course, in general).

The aesthetic system presented is decidedly "shapist". Interpretations of paintings are concerned mainly with shape. The basic component of a generative specification is a shape grammar. The aesthetic value assigned to an interpretation which refers to a painting is independent of the actual colors used in the painting. If two paintings differ only in the colors used, the aesthetic values assigned to their respective interpretations are equal.

Aesthetic viewpoints that emphasize color ("colorist" aesthetic viewpoints) are possible and, indeed, have achieved great popularity for non-representational, geometric paintings in the recent past. "Colorist" aesthetic systems for paintings definable by generative specifications could take several forms. A "colorist" aesthetic system could be defined that uses the same set of interpretations and reference decision algorithm as the aesthetic system described but that has a different evaluation function. The evaluation function could be based solely on the colors specified. As a very simple example, an aesthetic system could be defined with a decided preference for blue by having the aesthetic value assigned by the evaluation function be directly proportional to the percentage of the canvas painted in shades of blue. More interesting would be to define an aesthetic system that deals with the internal coherence of paintings definable by generative

specifications but that uses a set of interpretations and algorithm 4 based on color instead of shape. The α in an interpretation which refers to a painting could specify the structure and interrelationships of the colors used in the painting in terms of some color system. β in an interpretation which refers to a painting could specify the occurrences of the different colors in the painting. The evaluation function E_2 could be used. Certain combinations of colors would have simple specifications or generating functions α and if these occurred in a painting the interpretation could be assigned high aesthetic value. Random (in the sense of Kolmogorov) combinations of colors would have more lengthy specifications or generating functions (α) and their occurrence in a painting would necessarily result in interpretations assigned low aesthetic value.

Aesthetic systems with interpretations that deal with the external evocations of paintings are also possible. In such aesthetic systems, the first part of each interpretation that refers to a painting is a description of the painting. The second part is a specification of what is evoked by the painting. This specification could be in terms of images that are evoked, such as faces. Or, the emotions that are evoked, e.g. by the colors, could be specified. Another possibility is that the second part of the interpretation might specify the optical or three-dimensional perceptual effects of the painting. A painting based on the reversible figure of Section 1.5 might have an interpretation with high aesthetic value in a system like this. Aesthetic systems with interpretations that deal with the external evocations of paintings are easy to speculate about, but it seems difficult to really specify systems of this type that would reflect the interpretative conventions of people.

One difficult problem is constructing the algorithm A which given the description of the painting as input produces the evocations of the painting as output. What characterizes the area of a painting that a person sees as a face? What characterizes paintings that evoke a particular emotion in a person or that produces optical or three-dimensional effects? Why do viewers often perceive sexual or anatomical meaning in the painting Eve (Figure 30b)? These are difficult and weighty questions for which there are now, at best, only the vaguest of answers. I do believe that the rigor imposed by attempting to write algorithms that solve these problems can engender insights into the nature of these processes.

An intriguing possibility for a managable aesthetic system that deals with the external evocations of paintings definable by generative specifications is to single out shapes that have a symbolic meaning in our culture. For example, the paintings shown previously contain shapes such as yin-yang symbols, crosses, and stars. A list of some symbolic shapes and some description of their evocations (e.g. associations, meaning, attached emotions) would be made. The first part of an interpretation which refers to a painting would be a description of the painting (e.g. a shape, color, and occurrence table). The algorithm A in the aesthetic system would search through the description for symbolic shapes. The second part of an interpretation (i.e. the output of the algorithm) would be the evocations of any symbolic shapes found in the painting. If the evaluation function E_2 were used, interpretations which refer to paintings containing no symbolic shapes and hence no evocations would be assigned lowest aesthetic value. Interpretations of paintings with short descriptions and lengthy evocations would be assigned high aesthetic

value. This may or may not be desirable. For example, a swastika might have lengthy evocations and hence the interpretation of a painting containing a swastika might be assigned high aesthetic value by E_2. Assigning low (negative ?) aesthetic value to interpretations of paintings containing shape symbols with displeasing evocations would be possible with a different evaluation function.

The aesthetic system described in Section 3.2.5 could be combined with the aesthetic system just discussed using the previously mentioned method of composing an aesthetic system that deals with internal coherence with an aesthetic system that deals with external evocations. The first part of an interpretation in this aesthetic system would be a generative specification. The second part would be the evocations of symbolic shapes in the painting. The description of the painting would occur internally in the algorithm A which would be the composition of the algorithms of the two original aesthetic systems. Using E_2, high aesthetic value would be assigned to interpretations of paintings with short generative specifications and long evocations.

3.2.10 Aesthetic systems and science

> "... the concept of complexity might make it possible to precisely formulate the situation that a scientist faces when he has mede observations and wishes to understand them and make predictions. In order to do this the scientist searches for a theory that is in agreement with all his observations. We consider his observations to be represented by a binary string, and a theory to be a program that calculates this string. Scientists consider the simplest theory to be the best one, and that if a theory is too 'ad hoc', it is useless. How can we formulate these intuitions about the scientific method in a precise fashion? The simplicity of a theory is inversely proportional to the length of the program that constitutes it. That is to say, the best program for understanding or predicting observations is the shortest one that reproduces what the scientist has observed up to that moment. Also, if the program has the same number of bits as the observations, then it is useless, because it is too 'ad hoc'. If a string of observations only has theories that are programs with the samo length as the string of observations, then the observations are random, and can neither be comprehended nor predicted. They are what they are, and that is all; the scientist cannot have a theory in the proper sense of the concept; he can only show someone else what he observed and say 'it was this'.
>
> "In summary, the value of a scientific theory is that it enables one to compress many observations into few theoretical hypotheses. There is a theory only when the string of observations isn't random, that is to say, when its complexity is appreciably less than its length in bits. In this case, the scientist can communicate his observations to a colleague much more economically than by just transmitting the string of observations. He does this by sending his colleague the program that is his theory, and this program must have much fewer bits than the original string of observations." -- G. Chaitin [1973]

Aesthetic systems are applicable to the sciences as well as the arts. Aesthetic systems in science normally are concerned with internal coherence and can be considered to use the β form of the reference decision algorithm schema of Figure 46. The objects to which interpretations refer are the phenomena under

study. The sensory input transducer and linked algorithm of the reference decision algorithm correspond to the data collection mechanism. The description of the phenomena produced in the reference decision algorithm schema is the data. In an interpretation $\langle\alpha,\beta\rangle$ which refers to phenomena, β is the description of the phenomena or the data. α is a specification of the underlying structure of β. α is an encoding of the scientific laws or theory (and possibly some initial conditions) that specify/explain the data. The algorithm A embodies the mathematical conventions implicit in the theory. Given α as input, A produces β as output. Before proceeding further, it should be restated that aesthetic systems are designed to model the logical properties of a particular viewpoint and not the actual processes involved in using or applying a viewpoint. The attempt here is to describe the logical components and their interrelationships in a scientific aesthetic viewpoint and not the actual process of doing science. Aesthetic systems in science frequently use the evaluation function E_Z and associated order O_Z. Interpretations which refer to the same phenomena have identical β (data) and different α (scientific laws or theory). Applying the evaluation function E_Z to these interpretations, the shorter the length of α, the higher the assigned aesthetic value. The interpretation of phenomena which has the shortest explanation is assigned the highest aesthetic value. This is simply a restatement of Occam's razor or the law of parsimony, the traditional evaluative criteria of science (cf. [Rossi 1966]). This formulation of aesthetic systems for science is in full accord with Chaitin's [1973] discussion of the application of complexity theory to science quoted in part above. The everyday use of the words "beautiful" and "elegant" to describe mathematical

systems and physical laws is in this spirit -- parsimonious specification of seemingly complicated phenomena.

3.2.11 An aesthetic system implicit in Meta-Dendral

As an example of aesthetic systems in science, the Meta-Dendral system [Buchanan, et. al. 1971], a program for automatic theory formation in mass-spectrometry, embodies implicitly an aesthetic system. "The mass-spectrometer is an instrument which bombards molecules of a chemical sample with electrons and records the relative numbers of resulting charged fragments by mass ... Mass-spectrometry (MS) theory ... is a collection of statements about the fragmentation patterns of various types of molecules upon electron impact." The input to the Meta-Dendral program is "a large number of sets of data (fragment-mass tables) and the associated (known) molecular structures." The output of the Meta-Dendral program is "A set of ... rules constituting a subset of the theory of mass-spectrometry." (Quotes from [Buchanan et. al. 1971]). Given a chemical sample, the mass-spectrometer reports the fragment-mass table that results from bombarding the sample. The Meta-Dendral program is given the fragment-mass tables that are known to result from certain samples and outputs an inferred set of rules for the mass-spectrometry process.

Aesthetic systems could be formulated for mass-spectrometry in several ways. The most appropriate formulation at the level of Meta-Dendral is as follows:

α in an interpretation is some list of rules for mass-spectrometry. The algorithm A given the rules of α as input applies the rules to the known set of molecular structures and outputs as β the fragment-mass tables that would result. Thus for each interpretation $\langle\alpha,\beta\rangle$ in the set I_A, α is a list of MS rules and β is the fragment-mass tables that would result for the set of chemical molecular structures according to the MS rules of α. α is a theory; β is the predicted data. The β form of the reference decision algorithm schema of Figure 46 is used. S, the sensory input transducer and associated algorithm is the mass-spectrometer itself. The description λ (i.e. the output of S, the mass-spectrometer) is the actual fragment-mass tables that are produced by the mass-spectrometer for the chemical samples. For an interpretation $\langle\alpha,\beta\rangle$ to refer to the phenomena, β, the fragment-mass tables predicted by the rules of α, must be identical to λ, the fragment-mass tables that actually are produced by the mass-spectrometer. So, each interpretation $\langle\alpha,\beta\rangle$ in the set of interpretations I_A contains a possible MS theory (list of rules) as α and the predicted data as β. The interpretations which actually refer to the mass-spectrometry data are the interpretations in which β, the predicted data, is identical to λ, the actual data.

There may be many interpretations in I_A which refer to the phenomena, i.e. there may be many theories which predict the actual data. The evaluation function E_Z assigns high aesthetic value to interpretations with short α and long β. All interpretations which refer to the phenomena in this aesthetic system have identical β but different α. Thus, among the interpretations which refer to (account for) the phenomena, E_Z assigns highest aesthetic value to the interpretation with

the shortest α (shortest MS rules). Meta-Dendral actually does use an evaluation function similar to E_Z to choose among alternative theories that predict the data [Buchanan et. al. 1971].

The aesthetic system does not indicate the precise algorithms used by Meta-Dendral. Rather, it provides a logical framework in which the task of Meta-Dendral can be understood.

The Meta-Dendral problem can be restated as: Find a list of MS rules that predicts the fragment-mass tables that actually occurred. If there are more than one such lists of rules, find the simplest. Or, in aesthetic system terms: Find an α that produces a β which is identical to the description λ constructed in the reference decision algorithm. If there are more than one, find the simplest such α. This problem, a common one in science, is a special case of the general problem of analysis, which is investigated in the next section.

3.2.12 Aesthetic systems and analysis

An analysis problem arises when we try to understand an existing object as a work of art. The problem of analyzing an object in terms of a particular aesthetic viewpoint can be stated precisely using aesthetic systems : given an object and an aesthetic system $<I_A,R,E,O>$, find an interpretation in I_A which refers, using the reference decision algorithm R, to the object and which is assigned an aesthetic value by the evaluation function E which is maximal in the sense of the order O. This interpretation indicates the best way to understand the object from the aesthetic viewpoint specified by the aesthetic system.

Notice that this problem is different than the problem for which the computer program was designed. The program is given α (a generative specification) and asked to (1) use the algorithm A to construct the corresponding β, (2) evaluate this interpretation $<\alpha,\beta>$, and (3) construct an object (i.e. display a painting) to which the interpretation refers. In the analysis problem, we are given an object and asked to find an interpretation with high aesthetic value and which refers to the object. The analysis problem is considerably more difficult, as we shall see.

3.2.12.1 The analysis problem for aesthetic systems in general

The analysis problem for a given object and aesthetic system is to find an interpretation which refers to the object and which is assigned highest aesthetic value.

For a given aesthetic system and a given object, there is some (possibly empty) subset of the set of interpretations I_A which contains exactly the interpretations which refer to the object. The computability of this subset depends on the nature of the algorithm A and the reference decision algorithm R. If the algorithms A and R eventually must halt for every input, then the subset of interpretations which refer to an object (and the set I_A) is recursive. If the algorithms A and R need not halt for some inputs, then the subset of interpretations which refer to an object (and the set I_A) is only recursively enumerable. This latter case would make an analysis procedure much more difficult to implement as it would mean there would be no effective way of testing whether an interpretation refers to an object or even whether a pair of symbol strings is an interpretation in I_A. A more extensive discussion of the analysis problem from a computability point of view will be given in [Stiny, in preparation].

Theoretically, an analysis procedure is required to find the interpretation assigned highest aesthetic value which refers to the object. For an arbitrary evaluation function in an arbitrary aesthetic system, this is not always possible. One possible problem is that the subset of interpretations which refer to an object can be infinite. A possible practical solution to this problem is to set a threshold

and settle for any interpretation which refers to the object and is assigned aesthetic value above the threshold. For aesthetic systems using the evaluation function E_Z and containing interpretations with α and β defined over binary alphabets, a natural threshold might be 1 (see Section 3.2.4). A problem with this criteria is that an object might have an infinite number of interpretations which refer to it, none of which are assigned aesthetic value above the threshold. A better solution might be to only consider a finite subset of the set of interpretations and to choose the interpretation in that subset that refers to the object and is assigned highest aesthetic value. For example, only those interpretations where the length of each component is less than some very high, but fixed, value (i.e., $L(\alpha)<c$ and $L(\beta)<c$) might be considered.

An analysis procedure could have the structure of the following schema:

Step 1: Select a new interpretation.

Step 2: Does the interpretation refer to the object? If no, go to Step 5.

Step 3: Compute the aesthetic value for the interpretation.

Step 4: Is the computed aesthetic value the highest yet computed? If yes, save the interpretation and value.

Step 5: Halt? If yes, output the most recently saved interpretation and terminate. If no, go to Step 1.

The first four steps in this schema could be constructed using the four components of an aesthetic system. In Step 1 the set of interpretations is used, in Step 2 the reference decision algorithm, in Step 3 the evaluation function, and in Step 4 the

order. Of course, if the algorithms A and R might not terminate some allowance must be made for aborting a particular loop through the procedure so a new α can be tried.

The important part of an analysis procedure of this type is the first component, the procedure for selecting a new α. The selection procedure could be a simple enumerative scheme, but it would be much more efficient if the α were selected intelligently. Ideally, of course, the very first α selected would result in an interpretation which refers to the object and which is assigned highest possible aesthetic value and the program would know this and terminate immediately. This is usually infeasible. The sensory input transducer and parts of the reference decision algorithm could be attached directly to the selection procedure. It would be nice if the selection procedure reacted to the aesthetic values assigned previous interpretations and honed in on the interpretation assigned maximal aesthetic value. Use of this type of analysis procedure can be considered a search through a space of interpretations with a goal of finding the interpretation which refers to the object and which is assigned highest aesthetic value. The key is to have an intelligent selection procedure which makes extensive use of problem-specific knowledge (cf. [Feigenbaum et. al. 1971]).

An analysis procedure with the structure of the above schema is applicable to any type of aesthetic system. If some restrictions are put on the structure of components of an aesthetic system, the analysis problem sometimes is simplified. In particular, the analysis problem is simplified if the reference decision algorithm adheres to the reference decision algorithm schema of Figure 46.

3.2.12.2 The analysis problem for aesthetic systems with the α form of the reference decision algorithm schema

The analysis problem is trivial for an aesthetic system with a reference decision algorithm schema adhering to the α form of the schema in Figure 46. In this type of aesthetic system, only one interpretation can refer to a given object. This interpretation is $\langle\alpha,\beta\rangle = \langle\lambda,A(\lambda)\rangle$, where λ is the description of the object given internally in R.

3.2.12.3 The analysis problem for aesthetic systems with the β form of the reference decision algorithm

The analysis problem is extremely interesting for aesthetic systems with reference decision algorithms adhereing to the β form of the reference decision algorithm schema of Figure 46. In a reference decision algorithm of this type, the description λ of an object is produced internally by the algorithm linked to the sensory input transducer. By definition, this description is identical to the β of any interpretation $\langle\alpha,\beta\rangle$ which refers to the object. In a given aesthetic system of this type, there may be any number of interpretations which refer to a particular object, but all such interpretations must have identical β.

An analysis procedure for an aesthetic system of this typpe could have the structure of the following schema:

Step 1: Obtain the description λ of the object.

Step 2: Find a new sequence α that when used as input to the algorithm A produces λ as output.

Step 3: Compute the aesthetic value of the interpretation $<\alpha,\lambda>$.

Step 4: Is the computed aesthetic value the highest yet computed? If yes, save the interpretation and value.

Step 5: Halt? If yes, output the most recently saved interpretation and terminate. If no, go to Step 2.

First, the sensory input transducer and linked algorithm of the reference decision algorithm are used to obtain the description of the object. The next step is the interesting part of this procedure. The obtained description must, by definition, be identical to the β of any interpretation which refers to the object. So, β is known; the problem is to find the possible α. Again, a possible output β of the algorithm A is known; the problem is to find a possible input α that would produce that output. The construction of the second step of the analysis procedure involves the construction of the inverse of the algorithm A.

The problem of finding the inverse of an algorithm is of interest in its own right. McCarthy [1956] has investigated enumerative techniques for the inversion of the partial function defined by a Turing machine. At the simplest level, this consists of enumerating all possible tapes that can be formed using the input vocabulary, using each of these as input to the Turing machine to be inverted, and determining which of these input tapes produces the given output tape. The problem is that any individual computation might not terminate (i.e. the Turing machine defines a partial function). To remedy this, McCarthy proposes various

criteria for interleaving the computations. Upon investigating this problem, it became apparent that enumerating over all possible input tapes is unnecessary. A method was developed for beginning with an output tape and tracing backwards through the possible computations of the Turing machine to be inverted to obtain the possible input tapes. This method takes the form of a simple construction for obtaining a (non-deterministic) inverse Turing machine for any given Turing machine. Details of this construction are given in [Gips 1973] which is included as an appendix to this dissertation. Given an algorithm A (in the form of a Turing machine) the inverse algorithm A' can be constructed. Given β as input, A' outputs all possible α that could be used as input to A to produce β. That is, $A'(\beta) = \alpha$ iff $A(\alpha) = \beta$.

After an interpretation which refers to the object is constructed, it is evaluated using the aesthetic evaluation function E and this value is compared to the highest aesthetic value assigned previously. If it compares favourably, the new interpretation and its aesthetic value are saved. The process is repeated until some termination criteria are met.

Ideally, the first interpretation constructed for the object would be guaranteed to be aassigned highest possible aesthetic value. This is possible for certain evaluation functions and not possible for others. A procedure for obtaining input α from output β (based on either of the techniques discussed) can be designed so that the first α produced has certain properties. For example, it could be guaranteed that the first α produced is the input to the algorithm A that produces β in the fewest steps (i.e. in the shortest time). This would automatically

produce an interpretation which would maximize an evaluation function that assigns aesthetic value inversely proportional to the amount of time required to get β from α using A. Similarly, it could be guaranteed that the first α produced is the input to the algorithm A that requires the least extra memory (i.e. squares on the tape) to produce β. This would automatically produce an interpretation which would maximize an evaluation function that assigns aesthetic value inversely proportional to the amount of extra memory required to get β from α using A. These two evaluation functions correspond to the traditional time and space complexity measures defined for Turing machines [Hopcroft and Ullman 1969]. These evaluation functions are defined in terms of the computational resources required to get β from α using A. Unfortunately, there is no way of automatically finding an interpretation $<\alpha,\beta>$ guaranteed to maximize the evaluation function E_Z. Given an algorithm (that might not terminate) and an output, it is not always possible to find a sequence guaranteed to be the shortest input to the algorithm that would produce the output. The equivalent problem, which is also in general unsolvable, is given a non-deterministic algorithm and an input, find the shortest output. (If this problem were solvable, the construction in the Appendix could be used to solve the previous problem). The difficulty is that the shortest output might take an arbitrarily long amount of time to compute and might require an arbitrarily large amount of temporary memory.

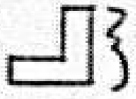
There is a guaranteed solution for the analysis problem for aesthetic systems of this type if the evaluation function is based on the amount of time or memory required in the computation of β from α using A. If an evaluation function such as E_Z is used, there is, in general, no guaranteed solution. Heuristics must be used.

3.2.12.4 The analysis problem for the aesthetic system for paintings definable using generative specifications

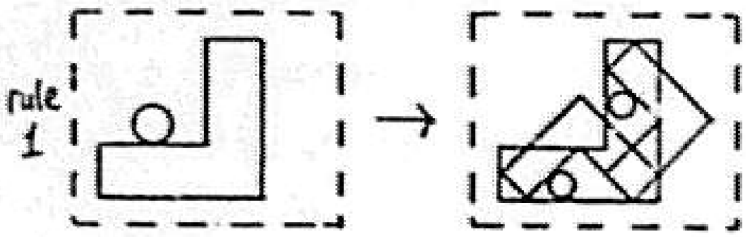
The analysis problem for the aesthetic system described in Section 3.2.5 for paintings definable by generative specifications is a special case of the problem just discussed. The problem is given a painting, find the shortest possible generative specification for the painting. From personal experience, this is a difficult problem. For example, the painting Anamorphism I (see Figure 49) was first generated using the generative specification shown in Figure 55. Only after repeated use of the painting was it realized that the shorter generative specification of Figure 50 also could be used. The length of the generative specification of Figure 55 (i.e. $L(\alpha)$) is 45 and the aesthetic value assigned the interpretation containing this generative specification is 5.64. This compares with a length of 41 and an aesthetic value of 6.20 using the generative specification of Figure 50. The difference is accounted for by the different terminal shapes: an "L" (which is composed of six straight lines) in the generative specification of Figure 55 versus a rectangle in the generative specification of Figure 50. The β (shape, color, and occurrence tables) produced by the two generative specifications are of course identical. This again illustrates that evaluation depends on interpretation, or, more specifically, that it is interpretations that actually are evaluated. Of course, it could be said that the highest aesthetic value found by an analysis procedure for an interpetation which refers to an object is the aesthetic value assigned to the object by the analysis procedure.

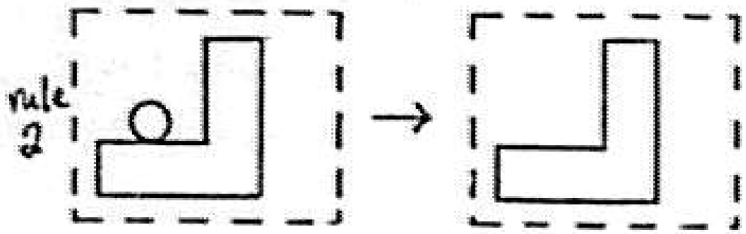
The problem of finding a generative specification for a painting is essentially

SHAPE GRAMMAR PSG19

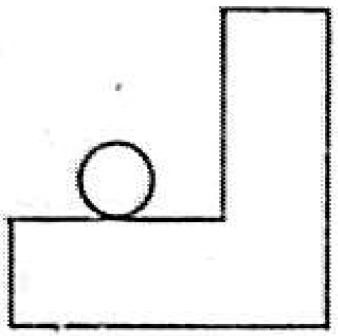
$V_T = \{$  $\}$ $V_M = \{$  $\}$

R contains





I is



SELECTION RULE: 2

PAINTING RULES

$L_2 \Rightarrow$ GREEN

$L_1 \cap \sim L_2 \Rightarrow$ RED

$L_0 \cap \sim (L_1 \cup L_2) \Rightarrow$ BLUE

$\sim (L_0 \cup L_1 \cup L_2) \Rightarrow$ LT. BLUE

LIMITING SHAPE

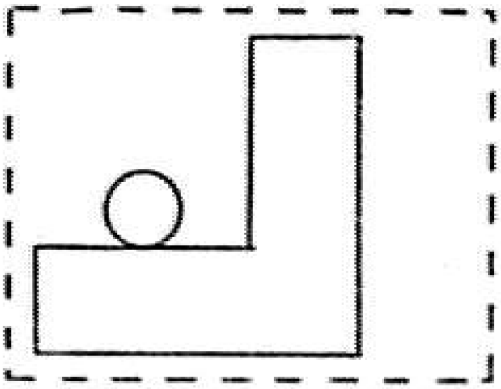


Figure 55. Original generative specification for Anamorphism I.

the problem of finding a shape grammar that generates the shapes that appear in the painting. In this sense, the problem is a version of the grammatical inference problem [Feldman et. al. 1969] [Biermann and Feldman 1972] [Feldman 1972] but for shape grammars instead of phrase structure grammars. Evans [1971] has written a program that infers a type of picture description grammar (see Section 1.8.3) given a set of descriptions of input pictures. Because the evaluation function E_Z is used in the aesthetic system, the problem is generally to find the shortest possible shape grammar that generates the shapes in the painting.

Finally, it should be noted that if the full range of shape grammars is allowed rather than the restricted type that can be defined using the computer program (see Section 3.2.1), then a generative specification can be trivially constructed for any painting. The shape grammar in such a generative specification would have one shape rule for each color (except the background) used in the painting. In the generation of the shape in the painting, each rule in the shape grammar is applied exactly once and results in the addition of all the shapes painted a particular color. This shape grammar would take into account none of the shape redundancy in the painting. A generative specification of this type would be relatively long and therefor the interpretation which includes this generative specification would be assigned minimal aesthetic value by E_Z. The shape, color, and occurrence tables that describe a painting that has no generative specification shorter than a trivial generative specification of this type can be considered random in the sense used in Section 3.2.4.

In the analysis of paintings in terms of interpretations containing the full range

of generative specifications, it is easy to construct an interpretation which refers to the painting and which is assigned minimal aesthetic value. It is difficult to find the interpretation which refers to the painting and which is assigned highest aesthetic value by E_Z.

3.2.13 Aesthetic systems and synthesis

A synthesis problem arises when we try to produce a new work of art. The synthesis problem for an aesthetic system is symmetric to the analysis problem and can be stated precisely : given an aesthetic system $\langle I_A, R, E, O \rangle$, construct an object for which there is an interpretation in I_A that refers, using the reference decision algorithm R, to the object and that is assigned aesthetic value by E that is maximal in the sense of the order O. The synthesis problem for aesthetic systems containing subsets of the set of interpretations in the aesthetic system described in Section 3.2.5 is be investigated here. A more general discussion of synthesis and aesthetic systems will appear in [Stiny and Gips 1974].

Suppose some constraints are put on allowable generative specifications. For example, the basic shape (see Section 3.1.2.1) might be required to be a rectangle, the rule shape might be required to consist of two instances of the basic shape, the initial shape might be required to consist of one instance of the basic shape, and the selection rule might be required to be 3. The problem is to find and display a painting with an interpretation that is assigned maximal aesthetic value

and that contains a generative specification with some given constraints. The set of interpretations containing generative specifications with some such constraints forms a subset of the set of interpretations in the aesthetic system of Section 3.2.5. This reduced set of interpretations together with the reference decision algorithm, the evaluation function (E_Z), and the order (O_Z) of the original aesthetic system can be considered to form a new aesthetic system. The problem of interest is the synthesis problem for such reduced aesthetic systems.

This problem can be divided into two parts. The first part is to find a generative specification with the given constraints which is part of an interpretation that is assigned highest possible aesthetic value. This can be considered the design problem for the aesthetic system. One possible approach to writing programs that perform this part of the synthesis problem is discussed in the next section. The second part is to actually construct and display the painting specified by the generative specification. Programs that perform this part of the synthesis problem are described in Section 3.1.2.

3.2.14 Design as search

> "All art presupposes a work of selection ... To proceed by elimination - to know how to discard, as the gambler says, that is the great technique of selection. And here again we find the search for the One out of the Many." -- Stravinsky [1947, p. 70]

A design program for paintings definable by generative specifications could be formulated as a search through a space of generative specifications. The goal of the search would be to find a generative specification from some restricted class that defines a painting that has an interpretation assigned maximal aesthetic value.

The space to be searched can be defined in terms of a subset of the generative specifications that are definable using the program and a collection of operators. Each point in the space is a generative specification. The operators transform one generative specification into another and thereby allow transitions between points in the space. The natural set of operators to use are the user commands for changing the terminal, rule, and initial shapes of the generative specification as described in Figure 35. A space of this type can be regarded as a state space [Nilsson 1971] for the design problem and can be conveniently represented as a directed graph. The nodes in the graph represent generative specifications. Arcs between nodes indicate allowable transitions using the operators.

A simple example of a space of generative specifications is shown in Figure 56. The space is composed of sixteen generative specifications and two operators. The nodes labelled N1 - N6 represent the generative specifications for

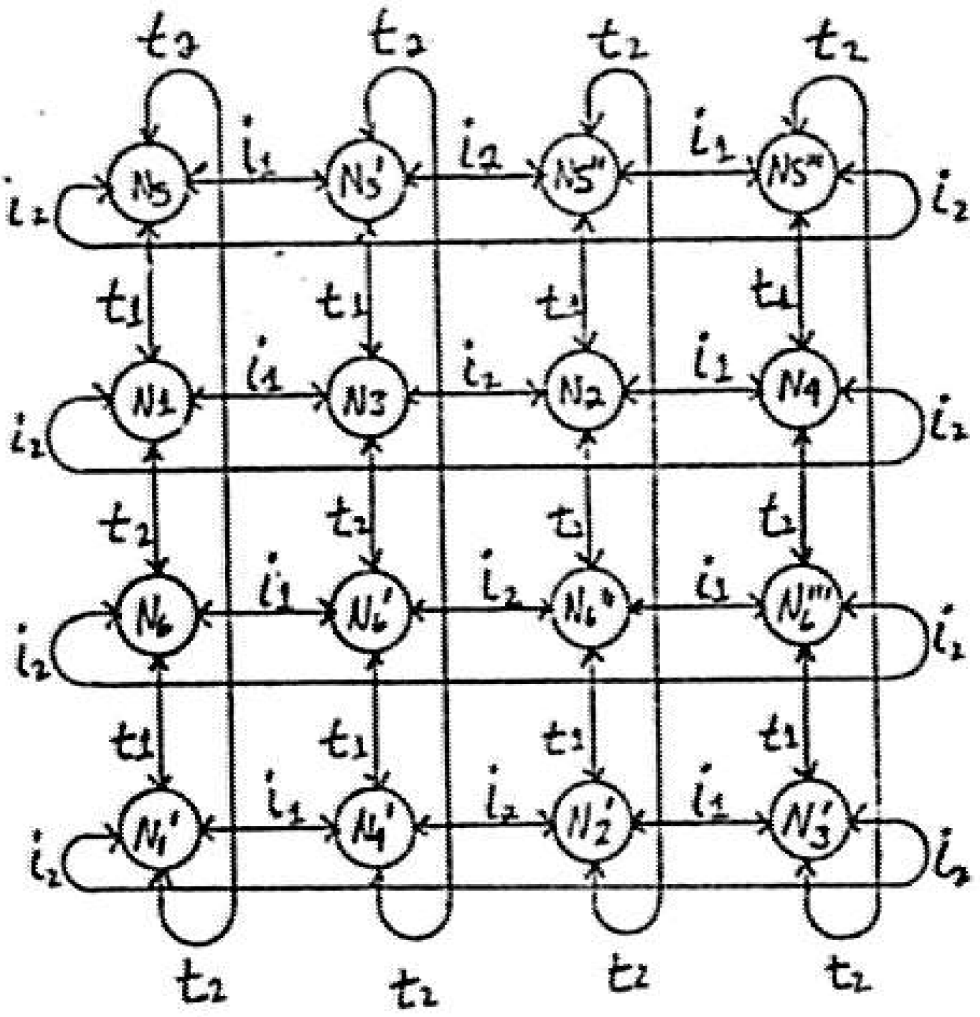


Figure 56. Space of sixteen generative specifications which contains generative specifications for Anamorphism I - VI.

Anamorphism I - VI shown in Figures 50 and 51. The other ten nodes represent variations of these generative specifications. The nodes labelled N1' - N4' represent generative specifications that specify paintings that are mirror images of Anamorphism I - IV. The nodes labelled N5' - N5"" and N6' - N6"" represent alternative generative specifications for Anamorphism V and Anamorphism VI respectively. The space thus contains sixteen points representing sixteen generative specifications that specify ten distinct paintings. In this example, there are severe constraints on allowable generative specifications. The sixteen generative specifications differ only in the locations of the markers in the right side of the first rule of the shape grammar (cf. Figure 51), i.e. the computer representations of the generative specifications differ only in the values of the θ and m parameters of the rule shapes (see Section 3.1.2.1). The two operators are turn (t) and invert (i), as defined in Figure 35 for the rule shapes. The arcs between the nodes in the figure show the transitions defined by applying the operators. The effect of applying t_1 to a generative specification is to rotate (turn) the first (left) shape of the rule shape 180 degrees (cf. the rule shape in Figure 36). Applying t_2 rotates the second (right) shape of the rule shape 180 degrees. Applying i_1 inverts the first shape of the rule shape. Applying i_2 inverts the second shape of the rule shape. The effects of these operators can best be seen by comparing the transitions in the space between points N1 - N6 with the shape rules for Anamorphism I - VI in Figures 50 and 51.

The space could be expanded (i.e. the constraints could be weakened) by allowing more of the commands for changing the terminal, rule, and initial shapes

specified in Figure 36 to be used as operators. For example, using the commmand turn 90 degrees as an operator instead of turn 180 degrees would double the size of the space. If operators were allowed for x and y translation of the rule shapes, the size of the space could be increased by a factor of thousands. Successively larger spaces could be obtained by allowing operators to change the scale of the rule shapes, to add or delete rule shapes, and finally to alter the terminal and initial shapes as well. If all the commands were allowed as operators, the space would contain all of the generative specifications definable using the program.

Spaces of generative specifications defined in this manner can be extremely large. Much computation time can be required to evaluate a given node (generative specification) in such a space. Because of this, exhaustive search through such spaces is not only undesirable but usually infeasible. The application of heuristics to guide the search is a possible remedy to this difficulty. Two types of heuristics would be helpful. Heuristics of the first type would restrict the number of nodes visited in the search, thereby precluding exhaustive enumeration. Good heuristics of this type would direct the search so that most visited points are associated with generative specifications likely to be solutions to the design problem. Heuristics of the second type would limit the amount of computation at visited nodes. Good heuristics of this type would halt computation at a visited node as soon as it became apparent that the generative specification associated with the node was unlikely to yield a solution to the design problem. Where heuristics can greatly reduce the amount of computation required for the search procedure to identify a solution to the design problem, the danger exists that the

heuristics may be inappropriate and may result in the search missing the best solutions.

A design procedure based on a heuristic search of a space of generative specifications could have the structure of the following schema:

Step 1: Select a new generative specification.

Step 2: Is this generative specification a suitable candidate? If no, go to Step 8.

Step 3: Generate the shapes to be painted.

Step 4: Is it worthwhile to continue with this generative specification? If no, go to Step 8.

Step 5: Construct the shape, color and occurrence tables associated with this generative specification.

Step 6: Compute the aesthetic value of this interpretation.

Step 7: Is the computed aesthetic value the highest yet computed? If yes, save the interpretation and value.

Step 8: Halt? If yes, output the most recently saved generative specification and terminate. If no, go to Step 1.

In the first step of this schema, a new generative specification to be investigated is selected. This selection procedure could be based on a depth-first search procedure, a breadth-first procedure, etc. [Nilsson 1971] combined with heuristics of the first type just described. Or, the selection procedure could be based on hill-climbing methods, e.g. the next generative specification is constructed by applying operators to the best generative specification found yet. The operators should be chosen heuristically to attempt to correct any identifiable

defects in that generative specification. Hopefully, the selection procedure would be intelligent enough to avoid regions of the space especially sparse in good generative specifications. At least the procedure should avoid regions of the space, e.g. the bottom row of the space in Figure 59, containing generative specifications which produce duplicates or mirror images of the paintings specified by other generative specifications.

Once a new generative specification is selected, heuristics of the second type would be used to determine whether the generative specification is a likely candidate. For example, if the generative specification obviously specifies a symmetric painting, it might be skipped as symmetric paintings tend to have interpretations assigned lower aesthetic value (see Section 3.2.9). It is expensive computationally to generate a painting from a generative specification. It is best to eliminate generative specifications with poor prospects as early as possible.

If the generative specification passes the initial tests, the shape grammar of the generative specification is used to generate the line drawing of the shapes to be painted. Once the line drawing is generated, new heuristics are used to estimate the aesthetic value that would be assigned. A crude estimate of the length of β might be the number of lines that have been generated. The generated line drawing might be required to be asymmetric to be investigated further. Again, investigating a node fully is expensive and the space is large. At each successive stage, more is known about the painting specified by the generative specification at the node. Nodes that have poor prospects of having high evaluations should be eliminated as soon as possible.

If the line drawing passes these tests, β (the shape color, and occurrence tables) is constructed and the interpretation is evaluated. If the assigned aesthetic value is the highest yet, the generative specification and aesthetic value are saved. The output of the design procedure is the best generative specification found. This generative specification can then be used to generate and display a painting using the programs described in Section 3.1.2.

A crude design procedure that would search through relatively small spaces could be implemented fairly easily given the programs that exist already. This would enable different search techniques and heuristics to be tried experimentally.

Research in the immediate future will be focused on automating the synthesis of paintings defined using generative specifications. A synthesis program would automatically construct and display a new painting which, given the constraints on allowable generative specifications, is guaranteed to have an interpretation assigned highest possible aesthetic value relative to the specified aesthetic system.

APPENDIX: A CONSTRUCTION FOR THE INVERSE OF A TURING MACHINE

[The problem of finding the inverse of an algorithm arose in connection with the analysis problem for a certain type of aesthetic system (see Section 3.2.12.3). A general and relatively efficient solution to this problem is described below. This appendix originally appeared as a Stanford Computer Science Report [Gips 1973]. Since this work was completed and the report appeared, I have learned that a similar construction technique was used by Fischer [1965] in proofs about classes of restricted types of Turing machines with multiple tapes. Bennett [1973] has recently reported interesting results on the related problem of the logical reversibility of computation.]

Problem

Given Turing machine M that for input tape I produces output tape O, i.e. M(I) = O, construct an inverse Turing machine M' such that M'(O) = I. M' should be non-deterministic so that the set of output tapes of M' given O is exactly the set of all possible input tapes to M that would produce O.

McCarthy [1956] has investigated enumerative techniques for the inversion of the partial function defined by a Turing machine. In this paper, a simple method for the direct construction of M' given M is presented.

Conventions

The Turing machine conventions used are basically those of [Minsky 1967].

A Turing machine is a set of quintuples of the form (old-state, symbol-scanned, new-state, symbol-written, direction) or (q_g, s_h, q_j, s_k, d). The states are $q_1, q_2, \ldots q_{n-1}$, halt, with q_b the initial state. The tape symbols are $s_1, s_2, \ldots s_m$. The directions are left, right, and " - ". Each quintuple is of one of three types:

(1) $(q_g, s_h, q_j, s_k$, left) interpreted as "if in state q_g and scan symbol s_h, then enter state q_j, write symbol s_k, and move tape head left".

(2) $(q_g, s_h, q_j, s_k$, right) interpreted as "if in state q_g and scan symbol s_h, then enter state q_j, write symbol s_k, and move tape head right".

(3) $(q_g, s_h$, halt, s_k, -) interpreted as "if in state q_g and scan symbol s_h, then write symbol s_k and halt in place".

An output tape of a Turing machine is a tape that exists after a quintuple of type 3 is applied.

Construction algorithm

Given Turing machine M with m tape symbols, n states, and p quintuples, a new Turing machine M' with m tape symbols, 2n states, and 2p+c quintuples, where c is the number of quintuples of M with the initial state as the first element, is constructed. The tape symbols of M' are the same as the tape symbols of M. If

the states of M are q_1, q_2, ... q_{n-1}, halt, the states of M' are begin, q_1', q_1'', q_2', q_2'', ... q_{n-1}', q_{n-1}'', halt. The initial state of M' is begin.

For each quintuple in M two quintuples are added to M' as follows:

(1) For each quintuple (q_g, s_h, q_j, s_k, left) of type 1 in M the quintuples (q_j', s_k, q_g', s_h, right) and (q_j', s_k, q_g'', s_h, left) are added to M'.

(2) For each quintuple (q_g, s_h, q_j, s_k, right) of type 2 in M the quintuples (q_j'', s_k, q_g', s_h, right) and (q_j'', s_k, q_g'', s_h, left) are added to M'.

(3) For each quintuple (q_g, s_h, halt, s_k, -) of type 3 in M, the quintuples (begin, s_k, q_g', s_h, right) and (begin, s_k, q_g'', s_h, left) are added to M'.

Additionally, one quintuple is added to M' for each quintuple in M that contains the initial state, q_b, as its first element (old-state) as follows:

(1) For each quintuple (q_b, s_h, q_j, s_k, left) of type 1 in M, where q_b is the initial state, the quintuple (q_j', s_k, halt, s_h, -) is added to M'.

(2) For each quintuple (q_b, s_h, q_j, s_k, right) of type 2 in M, where q_b is the initial state, the quintuple (q_j'', s_k, halt, s_h, -) is added to M'.

(3) For each quintuple (q_b, s_h, halt, s_k, -) of type 3 in M, where q_b is the initial state, the quintuple (begin, s_k, halt, s_h, -) is added to M'.

Description

The quintuples of M' are constructed by transposing the old-state and symbol-scanned of each quintuple of M with the new-state and symbol-written of that quintuple so that M' reverses the possible computations of M.

In the first part of the construction, the new-state and symbol-written of each quintuple of M becomes the old-state and symbol-scanned of two new quintuples of M'. If the new-state of the quintuple of M is halt, the old-state of the quintuples of M' is begin. If the direction of the quintuple of M is left, the old-state of the quintuples of M' is primed; if the direction is right, the old-state is double primed. The old-state and symbol-scanned of each quintuple of M becomes the new-state and symbol-written of the two added quintuples of M'. In one of the new quintuples of M' the direction is right and the new-state is primed; in the other the direction is left and the new-state is double primed. Informally, M' continually looks to the left and to the right to determine what quintuples of M could have been applied to result in the current state and tape configuration. The primes of the state of M' keeps track of which direction M' is currently looking.

In the second part of the construction, a quintuple with halt as the new-state is added to M' for each quintuple of M with the initial state as the old-state. Again, the new-state and symbol-written of the quintuple of M becomes the old-state and symbol-scanned of the quintuple of M'. If the new-state of the quintuple of M is halt, the old-state of the quintuple of M' is begin. If the direction of the quintuple of M is left, the old-state of the quintuple of M' is primed; if the

direction is right, the old-state is double primed. The symbol-scanned of the quintuple of M becomes the symbol-written of the quintuple of M'. Informally, an output tape of M' is produced at each point that M could have begun a computation.

M' is designed to trace backwards through all possible computations of M that could result in output tape 0. Each computation of M' has the same length as the corresponding computation of M. Further, the sequence of tape configurations for each computation of M' is identical to the reverse of the sequence of tape configurations for the corresponding computation of M, except for the locations of the tape heads. The sequence of states for each computation of M' is equivalent to the reverse of the sequence of states for the corresponding computation of M except for the first state of each sequence (which is always the initial state) and the final state of each sequence (which is always halt) if states q_x' and q_x'' in M' are considered equivalent to state q_x in M for $1 \leq x \leq n-1$.

Remarks

In general, the construction method does not result in quintuples in M' for every possible circumstance (i.e. every possible old-state : symbol-scanned pair). Some of the computations of M' may run into a dead-end, a situation where no quintuple is applicable. If an inverse machine M' does run into a dead-end, it simply means that M' is not retracing a computation that could have been done by

machine M. If dead-ends are displeasing then quintuples can be added to M' for every state : symbol pair not occurring as an old-state : symbol-scanned pair in M'. The new-state of these added quintuples would be an extra state that if entered results in M' looping forever. For practical use of the construction, dead-ends seem desirable as their use would increase the efficiency of a serial, deterministic encoding of M'.

Some of the computations of M' may not terminate. Recall that the goal for the construction is that the set of all output tapes of M' (i.e. all tapes that exist after M' enters the halt state) for input tape O is exactly the set of possible input tapes for M that would result in O as an output tape.

Example

As a simple example, consider Turing machine M_1 with four tape symbols: X, E, O, and b, where b is the symbol for "blank"; three states: q_1, q_2, and halt, where q_1 is the initial state; and four quintuples:

$(q_1, X, q_2, b, right)$

$(q_2, X, q_1, b, right)$

$(q_1, b, halt, E, -)$

$(q_2, b, halt, O, -)$

M_1 is a Turing machine that calculates the parity of the string of X's extending to

the right of the initial position of the tape head, erases the X's, and writes E if the parity is even and O if the parity is odd.

Sample input tape for M_i:

$$\ldots \; b \; b \; \underset{\uparrow}{X} \; X \; X \; X \; X \; b \; b \; \ldots$$

Resulting output tape:

$$\ldots \; b \; b \; b \; b \; b \; b \; b \; \underset{\uparrow}{O} \; b \; \ldots$$

The inverse Turing machine, M_i', has the same four tape symbols; six states: begin, q_1', q_1'', q_2', q_2'', and halt; and ten quintuples:

$(q_2'', b, q_1', X, right)$	from the first quintuple of M_i
$(q_2'', b, q_1'', X, left)$	from the first quintuple of M_i
$(q_1'', b, q_2', X, right)$	from the second quintuple of M_i
$(q_1'', b, q_2'', X, left)$	from the second quintuple of M_i
(begin, E, q_1', b, right)	from the third quintuple of M_i
(begin, E, q_1'', b, left)	from the third quintuple of M_i
(begin, O, q_2', b, right)	from the fourth quintuple of M_i
(begin, O, q_2'', b, left)	from the fourth quintuple of M_i
$(q_2'', b, halt, X, -)$	from the first quintuple of M_i
(begin, E, halt, b, -)	from the third quintuple of M_i.

The first eight quintuples result from adding two quintuples to M_i' for each quintuple in M_i. The final two quintuples result from adding one quintuple to M_i' for each quintuple in M_i with the initial state as its first element.

Sample input tape for M_i':

```
... b b b b b b b 0 b ...
                  ↑
```

Resulting output tapes:

```
... b b b b b b X b b ...
                ↑

... b b b b X X X b b ...
              ↑

... b b X X X X X b b ...
        ↑

        etc.
```

Applications

Where this investigation was motivated by a problem in a formal system for aesthetics, the construction seems widely applicable, e.g. in the theory of automatic programming. If M is a universal Turing machine then M' produces the possible input tapes and (specifications of) Turing machines that could produce a given

output tape. In particular, the construction can be used to obtain a Turing machine that produces (specifications of) all the possible Turing machines for a particular set of input tape : output tape pairs.

Anderson, R., "Syntax-directed recognition of handprinted two-dimensional mathematics" in M. Klerer and J. Reinfelds (eds.), Interactive Systems for Experimental Applied Mathematics, Academic Press, New York, 1968.

Arnheim, R. Art and Visual Perception, University of California Press, Berkeley, 1954.

Arnheim, R., Visual Thinking, University of California Press, Berkeley, 1969.

Arnheim, R., Entropy and Art, University of California Press, Berkeley, 1971.

Baumgart, B., "Winged edge polyhedron representation", Stanford Computer Science Report 320, October 1972.

Beardsley, M., Aesthetics, Harcourt Brace & World, New York, 1958.

Bennett, C., "Logical reversibility of computation", IBM Journal of Research and Development, November 1973, pp. 525-532.

Biermann, A. and J. Feldman, "A survey of results in grammatical inference", in S. Watanabe (ed.), Frontiers in Pattern Recognition, Academic Press, New York, 1972.

Birkhoff, G.D., Aesthetic Measure, Harvard University Press, Cambridge, Mass. 1933.

Blum, M. and C. Hewitt, "Automata on a two-dimensional tape", 8th IEEE Symposium on Switching and Automata Theory, 1967, pp. 155-160.

Brainerd, W., "Tree generating regular systems", Information and Control, Vol. 14, 1969, pp. 217-231.

Buchanan, B., E. Feigenbaum, and J. Lederberg, "A heuristic programming study of theory formation in science", Proceedings of the Second International Joint Conference on Artificial Intelligence, London, 1971.

Chaitin, G., "On the difficulty of computations", IEEE Transactions on Information Theory, Vol. IT-16, 1970, pp. 5-9.

Chaitin, G., "Some philosophical implications of information-theoretic computational complexity", SIGACT News, April 1973. Excerpted from "Information-theoretic limits of formal systems", Courant Institute Computational Complexity Symposium, New York, October 1971.

Chang, S.-K., "A method for the structural analysis of two-dimensional mathematical expressions", Information Sciences, Vol. 2, 1970, pp. 253-272.

Chang, S.-K., "Picture processing grammar and its applications", Information Sciences, Vol. 3, 1971, pp. 121-148.

Chomsky, N., "Three models for the description of language", I.R.E. Transactions on Information Theory, September 1956.

Chomsky, N., Syntactic Structures, Mouton & Co., The Hague, 1957.

Clowes, M., "Pictorial relationships - a syntactic approach", in B. Meltzer and D. Michie (eds.), Machine Intelligence 4, University Press, Edinburgh, 1969.

Dacey, M., "The syntax of a triangle and some other figures", Pattern Recognition, Vol. 2, Jan. 1970, pp. 11-31.

Dacey, M., "POLY: a two-dimensional language for a class of polygons", Pattern Recognition, Vol. 3, July 1971, pp. 197-208.

Duda, R. and P. Hart, Pattern Classification and Scene Analysis, Wiley, 1973.

Eastman, C. and C. Yessios, "An efficient algorithm for finding the union, intersection, and differences of spatial domains", Computer Science Report, Carnegie-Mellon University, September 1972.

Eden, M., "Handwriting generation and recognition", in P. Kolers and M. Eden (eds.), Recognizing Patterns, The M.I.T. Press, Cambridge, Mass., 1968.

Evans, T., "A grammar-controlled pattern analyzer", Proceedings of IFIP Congress 68, North Holland Publishing Co., Amsterdam, 1969.

Evans, T., "Grammatical inference techniques in pattern analysis", in J. Tou (ed.), Software Engineering 2, Academic Press, New York, 1971.

Feder, J., "Plex languages", Information Sciences, Vol. 3, 1971, pp. 225-241.

Feigenbaum, E., B. Buchanan, and J. Lederberg, "On generality and problem solving: a case study using the Dendral program", in B. Meltzer and D. Michie (eds.), Machine Intelligence 6, American Elsevier, New York, 1971.

Feldman, J., "Some decidability results on grammatical inference and complexity", Information and Control, Vol. 20, No. 3, April 1972, pp. 244-262.

Feldman, J., J. Gips, J. Horning, and S. Reder, "Grammatical complexity and inference", Stanford Computer Science Report 125, June 1969.

Feldman, J. and P. Rovner, "An Algol-based associative language", Communications of the ACM, Vol. 12, No. 8, Aug. 1969, pp. 439-449.

Fischer, P., "Turing machines with restricted memory access", Information and Control, Vol. 9, 1966, pp. 364-379.

Firschein, O., M. Fischler, L.S. Coles, J.M. Tenenbaum, "Forecasting and assessing the impact of Artificial Intelligence on society", Third International Joint Conference on Artificial Intelligence, Stanford, 1973.

Focillon, H., The Life of Forms in Art, Wittenborn Schultz, New York, 1948.

Frye, N., Anatomy of Criticism, Princeton University Press, Princeton, N.J., 1957.

Fu, K.-S. and B. Bhargava, "Tree systems for syntactic pattern recognition", presented at the Computer Image Processing and Recognition Symposium, University of Missouri-Columbia, 1972.

Fu, K-S. and P. Swain, "On syntactic pattern recognition", in J. Tou (ed.), Software Engineering, Academic Press, New York, 1971

Gardner, M., "The fantastic combinations of John Conway's new solitaire game 'life'", Scientific American, Oct. 1970, pp. 120-123.

Gips, J., "A new reversible figure", Perceptual and Motor Skills, Vol. 34, 1972, p.306.

Gips, J., "A construction for the inverse of a Turing machine" Stanford Computer Science Report 390, August 1973.

Gips, J. and G. Stiny, "Aesthetics systems", Stanford Computer Science Report 339, January 1973.

Gips, J. and G. Stiny, "An investigation of algorithmic aesthetics", submitted for publication.

Gombrich, E., Art and Illusion, Princeton University Press, Princeton, N.J., 1960.

Gombrich, E., "Expression and communication", in Meditations on a Hobby Horse, Phaidon Press Ltd., London, 1963.

Goodman, N., Languages of Art, Bobbs-Merrill, Indianapolis, 1968.

Guzman, A., "Decomposition of a visual scene into three-dimensional bodies", Proceedings Fall Joint Computer Conference, Vol. 33, 1968.

Hahn, H., "Geometry and intuition", Scientific American, April 1954.

Hopcroft, J. and J. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, Reading, Mass., 1969.

Huffman, D., "Impossible objects as nonsense sentences", in B. Meltzer and D. Michie (eds.), Machine Intelligence 6, American Elsevier, New York, 1971.

Kasner, E. and J. Newman, Mathematics and the Imagination, G. Bell and Sons, Ltd., 1949, 1965.

Kirsch, R., "Computer interpretation of English text and picture patterns", IEEE Transactions on Electronic Computers, Vol. EC-13, August 1964, pp. 363-376.

Kolmogorov, A.N., "Logical basis for information theory and probability theory", IEEE Transactions on Information Theory, Vol. IT-14, No. 5, 1968.

Ledley, et. al., "FIDAC: film input to digital automatic computer and associated syntax-directed pattern recognition programming system", in J. Tippep, et. al. (eds.), Optical and Electro-Optical Information Processing, The M.I.T. Press, Cambridge, Mass., 1965.

Luckiesh, M., Visual Illusions, Dover, New York, 1922, 1965.

Maggiolo-Schettini, A. "The theory of Markov-like algorithms with applications to picture processing", I.B.M. Research Report RC 4172, I.B.M., Yorktown Heights, New York, 1973.

Martin-Lof, P., "The definition of random sequences", Information and Control, Vol. 9, 1966, pp. 602-619.

McCarthy, J., "The inversion of functions defined by Turing machines", in C. Shannon and J. McCarthy (eds.), Automata Studies, Princeton University Press, Princeton, N.J., 1956.

Mercer, A. and A. Rosenfeld, "An array grammar programming system", Communications of the ACM, Vol. 16, No. 5, May 1973, pp. 299-305.

Menninga, L., "A syntax-directed approach to pattern recognition and description", in Proceedings Fall Joint Computer Conference 1971, pp. 145-151.

Meyer, L., "Some remarks on value and greatness in music", Journal of Art Theory and Criticism, 1959. Reprinted in M. Beardsley and H. Schueller (eds.), Aesthetic Inquiry, Dickenson Publishing, Belmont, Cal., 1967.

Milgram, D. and A. Rosenfeld, "A note on grammars with coordinates", Computer Science Center Report 70-140, University of Maryland, September 1970.

Milgram, D. and A. Rosenfeld, "Array automata and array grammars", Proceedings IFIP Congress 71, North Holland Publishing Co., Amsterdam, 1972.

Miller, W. and A. Shaw, "Linguistic methods in picture processing - a survey", Proceedings of the Fall Joint Computer Conference, 1968.

Minsky, M., Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, N.J., 1967.

Montanari, U., "Separable graphs, planar graphs, and web grammars", Information and Control, Vol. 16, May 1970, pp. 243-267.

Moore, E.H., "On certain crinkly curves", Transactions of the American Mathematical Society, vol. 1, pp. 72-90, 1900.

Munari, B., Discovery of the Square, George Wittenborn, Inc., New York, 1965.

Mylopoulos, J., "On the relation of graph grammars and graph automata", 13th IEEE Symposium on Switching and Automata Theory, 1972, pp. 447-470.

Narasimhan, R., "Syntax-directed interpretation of classes of pictures", Communications of the ACM, Vol. 9, No. 3, March 1966, pp. 166-173.

Narasimhan, R., "Picture languages", in S. Kaneff (ed.), Picture Language Machines, Academic Press, 1970.

Newman, W. and R. Sproull, Principles of Interactive Graphics, McGraw-Hill, New York, 1973.

Nilsson, N., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

Pavlidis, T., "Linear and context-free graph grammars", Journal of the ACM, Vol. 19, No. 1, January 1972, pp. 11-22.

Peano, G., "Sur un courbe, qui remplit toute un aire plane", Mathematische Annalen, Vol. 36, pp. 157-160, 1890.

Pfaltz, J., "Web grammars and picture description", Computer Graphics and Image Processing, Vol. 1, No. 2, August 1972, pp. 193-220.

Pfaltz, J. and A. Rosenfeld, "Web grammars", Proceedings of the First International Joint Conference on Artificial Intelligence, Washington, D.C., 1969, pp. 609-619.

Rosenfeld, A., "Isotonic grammars, parallel grammars, and picture grammars", in B. Meltzer and D. Michie (eds.), Machine Intelligence 6, American Elsevier, New York, 1971.

Rosenfeld, A., "Progress in picture processing : 1969-71", Computing Surveys, Vol. 5, No. 2, June 1973a, pp. 81-108.

Rosenfeld, A., "Array grammar normal forms", Information and Control, Vol. 23, No. 2, September 1973b, pp. 173-182.

Rosenfeld, A. and D. Milgram, "Web automata and web grammars", in B. Meltzer and D. Michie (eds.), Machine Intelligence 7, John Wiley & Sons, New York, 1972.

Rosenfeld, A. and J. Strong, "A grammar for maps", in J. Tou (ed.), Software Engineering, Vol. 2, Academic Press, New York, 1971, pp. 227-239.

Rossi, B., "The esthetic motivation in science", in G. Kepes (ed.), The New Landscape in Art and Science, Paul Theobald and Co., Chicago, 1966.

Schrandt, R. and S. Ulam, "On recursively defined geometric objects and patterns of growth", Los Alamos Scientific Laboratory Report LA-3762, Los Alamos, New Mexico, 1967.

Schwebel, J., "A graph-structure transformation model for picture processing", Department of Computer Science Report R-72-514, University of Illinois at Urbana-Champaign, May 1972.

Shannon, C. and W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana, Illinois, 1949.

Shaw, A., "A formal picture description scheme for picture processing systems", Information and Control, Vol. 14, No. 1, January 1969, pp. 9-52.

Shaw, A., "Parsing of graph-representable pictures", Journal of the ACM, Vol. 17, No. 3, July 1970, pp. 453-481.

Shaw, A., "Picture graphs, grammars, and parsing", in S. Watanabe (ed.), Frontiers of Pattern Recognition, Academic Press, New York, 1972, pp. 491-510.

Shepard, R., and J. Metzler, "Mental rotation of three-dimensional objects", Science, Vol. 171, 19 February 1971, pp. 701-703.

Sirmoney, G., R. Sirmoney, and K. Krithivasan, "Picture languages and array rewriting rules", Information and Control, Vol. 22, No. 5, June 1973, pp. 447-470

Smith, A.R. III "Two-dimensional formal languages and pattern recognition by cellular automata", 12th IEEE Symposium on Switching and Automata Theory, 1970, pp. 216-224.

Solomonoff, R., "A formal theory of inductive inference", Information and Control, Vol. 7, 1964, pp. 1-22 and 224-254.

Stiny, G., "Mathematical aspects of formal models in the arts", Ph.D. thesis, Systems Science Department, U.C.L.A., in preparation.

Stiny, G. and J. Gips, "Shape grammars and the generative specification of painting and sculpture", Proceedings of IFIP Congress 71, North Holland Publishing Co., Amsterdam, 1972. Also, in O. Petrocelli (ed.), The Best Computer Papers of 1971, Auerbach, Inc., 1972.

Stiny, G. and J. Gips, "Formalization of analysis and design in the arts", invited paper, Symposium on Basic Questions of Design Theory, Columbia University, May 1974. Proceedings to be published by North Holland Publishing Co., Amsterdam.

Stravinsky, I., Poetics of Music, Random House, New York, 1947.

Uhr, L., "Flexible linguistic pattern recognition", Pattern Recognition, Vol. 3, 1971, pp. 363-383.

VanLehn, K., (ed.), "SAIL user manual", Stanford Computer Science Report 373, July 1973.

Waltz, D., "Generating semantic descriptions from drawings of scenes with shadows", M.I.T. A.I. Memo TR-271, November 1972.

Williams, K., "Syntactic pattern recognition via unordered tree automata", Department of Computer Science Report TR 73-01, Michigan State University, 1973.