A System for Teaching the Classification of Geometric Patterns

V. V. Maksimov

ca. 1975; translation from the Russian: M. Eskina (2003)

1. Introduction

The present article offers a description of a system modeling the capacity of people to find a principle of classification (a distinguishing rule) of certain geometric objects, having only a small number of samples. The necessary requirement is that the system find the same principle of classification (among a number of a priori possible principles) which is found in the same problem by people. Such a requirement has naturally led to the following principles of system organization: 1) the language of the system describing the distinguishing rule should be terminologically akin to the language used in the same problems by a person; 2) since the same language can be used to describe several distinguishing rules applicable to the given input material, distinguishing rules in the system should be used in the order close to the order in which a person uses them.

Lately, a so-called "structural" or "linguistic" approach to problems of pattern recognition has gained popularity [1]. Basically, this approach presupposes describing objects in terms of separate typical details or fragments. For the description of objects a "language" is created whose "words" are fragments and whose grammar is represented by the possible ways of combining those fragments. However, this approach meets with several difficulties. The first difficulty is the choice of the dictionary. Any dictionary of fragments chosen *a priori* would be applicable only to a very small number of problems. Such a dictionary lacks the main characteristic of human language: the same word (for example, "big" or "small" picture, "angle," "inside," etc.) in different problems (and sometimes even in different pictures in the same problem) has different meanings. To a certain extent this difficulty can be overcome through prior training [2]. It is obvious, however, that a dictionary of fragments formed a priori will not be successful in all problems: the hope of prior training is based on the supposition that the variety of patterns in a single problem is much smaller than their variety in the sum total of all the problems. The second difficulty of the structural approach lies in the fact that in pattern-recognition problems the classification principle often cannot be described in terms of local characteristics of pictures, typical fragments, etc. Finally, the linguistic approach (at least at the present stage of its development) cannot offer any satisfying methods that could help search for such a pattern description.

As a system that provides a possibility of a complicated and deep search, we should mention a program created by T. Evans [3]. This program models human behavior while solving complicated visual test-problems. Such tests are formulated as follows: "Two pictures are given; between them there exist certain correlations. Find analogous correlations between a third picture and one of five other pictures given." Just as in the case of recognition problems, the evaluation of the test results is not defined by any concrete, strict rule, but is based on the comparison with the results of the corresponding experiment made on humans of admittedly high mental capacities. The behavior and results obtained by this program in such tests are comparable with those of humans.

Regrettably, Evans's work almost completely ignores the problem of restricting the search — cutting off branches in the decision process; it is mostly adjusted to the specifics of the problem domain. Firstly, the language used to compare geometric patterns in these tests does not have to contain non-discrete characteristics, because the ready-made (on the basis of such non-discrete characteristics) predicates are sufficient: to the left, to the right, more, less, etc. This greatly restricts the number of variants to be analyzed. Second, in a problem with a choice from a fixed number of answers, the question of order of search becomes meaningless. Indeed, instead of choosing just one answer from the multitude of different answers formulated in a given language and satisfying the conditions of a certain problem — an answer that would be the most natural from a person's point of view — in this case it is sufficient to choose one out of five given pictures, satisfying the necessary conditions. On the contrary, recognition training is not a multiple-choice problem[1].

A detailed analysis of the difficulties concerning the search for a description principle for classifying geometric patterns is given in the book by M. Bongard [5]. Bongard's book describes a project of a system that, as it seemed to us, is capable of overcoming these difficulties. A more concrete and detailed description of this project can be found in [6]. In [7] and [8] consecutive (initial) variants of implementation of this system (as a computer program) and experiments on training can be found. The present article provides a description of algorithmic work in separate blocks, as well as of the entire system. In addition, description and analysis of several experiments is given.

This program is designed for solving problems in which people describe distinguishing rules in strictly geometric terms: picture, line, part of picture, contour, area, length, slope, angle, subset, etc. The input objects for the program are flat black-and-white pictures given on a 45cm × 64cm raster. For training, the machine receives an input of several pictures divided into two classes. As a result of analyzing these pictures, the program of classification training is expected to formulate a rule for distinguishing one class of pictures from another.

2. Objects and Operators

2.1. Basic definitions. Pictures belonging to the training set, as well as all the intermediate results of analysis, will be called objects. There are three types of objects: pictures, numbers, and Boolean numbers (0 and 1). Transformations of objects into other objects are done by operators. The objects obtained on the output of one operator can be inputted to another operator. Such a superposition of two or more operators will also be an operator.

Let us define a collection of objects. A collection of training pictures is a collection of objects. A sum total of objects received at the output of an operator as a result of applying this operator to all the objects of a certain collection will also be a collection of objects. We will distinguish collections of pictures, collections of numbers and collections of

[1] We must notice that the initial ideas of the director of this research, M. Minsky [4], concerned application of this "language of description of correlations among figures" precisely for pattern recognition. Later the authors modified the problem.

Boolean numbers[2]. The concept of a collection of objects is convenient, on one hand, because a program never works on a sole object but always on their collections. On the other hand, it allows us to broaden the concept of operator. In the next section operators not applicable to separate objects will be described. Arguments and results of working of such operators can be only collections of objects.

Each operator establishes a correspondence between certain objects of the input and output collections. For instance, let us say that a given outlined picture corresponds to some other picture (from which the first picture was obtained by using the operator "contour isolation"), or that a certain set of pictures (or their parts) corresponds to a picture that was transformed (by the use of the operator "breaking into parts") into this set of pictures. If a certain operator puts every single object of the input collection into correspondence with a different single object of the output collection we will speak of a one-to-one correspondence. In the general case the correspondence among the objects of the collections does not have to be one-to-one.

Two more remarks about the operators need to be mentioned: 1) operators can have several inputs (every one of which has a separate collection of objects as input data); 2) operators can have several outputs. Operators with several outputs are formally indistinguishable from a set of several operators each one having a single output. However, since constructively such a set represents a unity, we will speak of a single operator with several outputs.

We will call *distinguishing collection* a collection of Boolean numbers in which 1) Boolean numbers are in one-to-one correspondence with the pictures of the training collection and 2) the Boolean number 1 corresponds to each picture of one class and the Boolean number 0 corresponds to each picture of the other class.

2.2. Notation.

1. Individual objects are represented by lowercase Roman letters: *p*: pictures; *n*: numbers; *b*: Boolean numbers.

2. Collections of objects (of each of the three types) are represented by the corresponding uppercase letters: *P*, *N*, *B*.

3. Indices: a) to distinguish collections of the same type among themselves, bottom indices are used: P_1, N_φ. Analogously, bottom indices are used to distinguish individual objects of the same type that belong to different collections: p_i, b_c, $n_s \in N_s$ ...; b) to distinguish objects of the same collection among themselves, top indices are used: $b_k^4 \in B_k$, $p^{ij} \in P$, ...

4. Operators are represented by uppercase letters (with the exception of *P*, *N*, *B*). The application of an operator to an individual object is written as left product: Dp, $M_k p$, $N = QB$, ...

Some auxiliary operations, found in the description of the structure of elementary operators in §3, are organized by analogy with elementary operators even though they

[2] Strictly speaking, the above definition of a collection of objects as a sum total (without taking into account its internal structure) is correct only for collections of the first-level of application. The definition of the level of objects, as well as a more complete definition of a collection of objects will be given in §3.5.

themselves are not such operators (they are not included in the list of elementary operators). They also transform objects into objects. An example of such an operation is the spreading of picture, *W*. For such operations we will use regular notation: $p_1 = W_p$.

5. As it has already been said, operators can have several (more than one) inputs and outputs. In such case we will represent the input and the output collections of objects by an ordered sequence of corresponding symbols inside round parentheses (*N*, *B*), $P_1 = U(P, B)$, $(B_1, B_2, ..., B_k) = H(N, B_0)$; we will represent in the same way the set of individual objects $\left(b^i_{la}, b^i_{lb}, b^i_{lc}, b^i_{ld}\right) = Lb^{ij}_m$.

6. In case a certain operator puts into correspondence to each object of the input collection one and only one object of the output collection, the indices will remain the same: $p^k_1 = Dp^k$, $n^{ij}_x = Xp^{ij}$.

7. We will say that a certain collection of objects *G* ($g_i \in G$) *characterizes* a collection of pictures *P* ($p^i \in P$) if there exists an operator *O* such that $g^i = Op^i$. We will also say in this case that an individual object g^i characterizes the given picture p^i.

8. *Subsets of the objects of a collection.* Let us have a collection of Boolean numbers *B* and a certain collection of objects *G* such that among the objects of these collections ($b^i \in B$, $g^i \in G$) there exists a one-to-one correspondence. Let the subset $G_B \subset G$ contain such and only such objects $g^i \in G$ that the corresponding Boolean numbers $b^i \in B$ equal 1. In this case we will say that the collection of Boolean numbers *B defines the subset of objects* G_B of the collection *G*.

2.3. Objects and collections of objects. *Additional definitions, machine implementation.* Pictures are given on a rectangular 45 × 64 raster. Points (elements) of raster α_{ij} (i = 1, 2,…, 45; j = 1, 2,…, 64) may be in two states, 0 and 1. We will call elements in state 1 “black” (excited), and elements in state 0 “white” (empty or non-excited).

A *picture p* will be the totality of excited points of the raster. A picture that does not contain excited elements and corresponds to the empty picture will be denoted by ϕ.

We will say that an element is *added* to a picture if this element passes into an excited state. An element is *erased* from a picture if this element passes into a non-excited state. The description of pictures in terms of sets of excited points of the raster will allow us to use operations, while describing algorithms of operators processing these pictures: union $A \cup B$, intersection $A \cap B$, complement $\overline{A}$, difference $A \setminus B$ and symmetrical difference $A \triangle B = (A \setminus B) \cap (B \setminus A)$, as well as correlations between pictures ($=$, $\subseteq$), all those in terms of set theory.

Let us define *neighboring* points of the raster. We will use two variants of the neighboring position. We will call two points of the raster $\alpha_{i_1 j_1}$ and $\alpha_{i_2 j_2}$ neighboring points if:

1) $|i_1 - i_2| \le 1$, $|j_1 - j_2| \le 1$;

or

2) $|i_1 - i_2| + |j_1 - j_2| \le 1$.

Every point of the raster (except the border points) is surrounded by eight neighbors in the sense 1 (within the area 3×3, with the center at the point) and by only four neighbors in the sense 2.

We will say that picture p is *connected* if for any pair of points (α, ω) $\alpha \in p$, $\omega \in p$, there exists a sequence of points

$$\alpha = \beta_0, \beta_1, ..., \beta_l, \beta_{l+1} = \omega, \ \beta_i \in p, \ i = 1, 2, ..., l,$$

such that each pair of points (β_i, β_{i+1}), $i = 1, 2, ..., l$ is a pair of neighboring points. In this way two variants of the neighboring position of the points of the raster define two variants of connectedness of pictures.

A *column of the raster* ξ_j is the sum total of the elements of raster α_{ij}, where j is fixed, and i goes through all its values ($i = 1, 2, ..., 45$). In the machine implementation, one memory cell of the M-20 machine corresponds to one column of the raster.

Collections of pictures, as already mentioned, are sets of individual pictures. Let us suppose that the pictures in collections are ordered. The top index in the notations p^5, $p^j \in P$ corresponds to the ordinal number of the picture in the collection. In the same way objects in collections of numbers and Boolean numbers are ordered.

Such a description of collections in the form of ordered sets of objects is customary for the machine implementation and, on the other hand, convenient for describing the structure of some of the program blocks. However, the order of objects in those sets is not essential for the functioning of the program. For instance, the result of program training does not depend on the order of pictures in the problem, i.e., on the order in which these pictures are inputted into the machine's memory.

Numbers — objects of the second type — are written in six binary digits. As shown by practice, such (rough) approximation does not affect the sum total of the problems to solve and does not appreciably narrow the area of program application. In addition, it allows a substantial saving of memory (making it possible to place seven numbers in one memory cell) and time.

Some of the operators that give numbers on the output may sometimes not be applicable. For example, a round picture does not have any slope, thus the measuring operator slope of the picture (see page 43) in this case cannot give any numerical output. In order to mark such "non-measurable" values, one of these six-digit codes — namely, 111 111 — is used. We will denote it by the letter *f*. Consequently, numbers can take on values $n = 0, 1, ..., 62, f$.

Collections of numbers can be obtained on the output of some elementary operators as a result of applying these operators to other collections of numbers. In this way operators define the "dimensions" of collections of numbers on the output. We will say that the collections of numbers have the same dimension if they are obtained by applying the same elementary operator. Otherwise collections of numbers will have different dimensions.

2.4. General structural scheme of the program. The program contains a certain number of elementary operators and combining rules that allow it to build complex operators. The goal of the program while solving a given problem is to build a complex

operator which, applied to a collection of training pictures, gives a distinguishing collection of Boolean numbers as output. The structural description of such a complex operator (in terms of elementary operators) will be the description of the distinguishing rule in the language of the program.

The totality of primitive operators defines the set of distinguishing rules that can be described in this language in principle. Thus, the first issue that should be confronted while creating a program is the choice of an adequate collection of primitive operators.

For every problem that interests us it is necessary that there exists a distinguishing rule described in terms of these primitive operators. Moreover, it is required that this distinguishing rule coincides (not only in the training material but in the entire set of pictures that people classify as belonging to a certain class) with the principle of classification that is found in the same problem by people. On the other hand, the program in the solution process, when presented with a collection of training pictures, must be able to find this and precisely this distinguishing rule. Thus, the second problem to be solved is the organization of the effective search for the distinguishing rule.

Since a complex operator can be built via consecutive application of primitive operators to the objects obtained on the output of other operators, the general scheme of the work of this program is the following: All the obtained object collections are kept in machine's memory. On each work-cycle of the program, a primitive operator is applied to one of these collections, producing on the output new collections of objects. These collections are also saved in machine's memory. Thus, in the beginning there exists only the initial collection of training pictures; then, in the process of work, the number of object collections in memory grows until one of the collections of Boolean numbers recorded in memory proves to be the distinguishing one.

For each collection of objects appearing in machine's memory it is known which one of the elementary operators was applied and which one of the input collections was transformed to obtain the given collection. This allows us to recreate the "sequence" of the elementary operators that led to obtaining a distinguishing collection from the collection of training pictures, that is, it allows us to recreate the unknown structure of the complex operator applied.

Thus the capacity of the program (the set of problems which the program is able to solve) is defined by the totality of primitive operators and the order of their consecutive execution. The structure of primitive operators is described in §3, while the order of their execution in §4.

It must be noted that in creating the given program the goal was not to find all of the primitive operators necessary to solve a wide variety of problems (for example, all of the problems from [5]). We have tried to find only the basic types of operators rather than a complete collection of operators of each type and to build corresponding schemes of consecutive application for them. In consequence, lists of some types of primitive operators (for instance, drawing and measuring operators, as well as operators breaking pictures into parts — see the following section) could be substantially enlarged and the spectrum of solvable problems would be broadened without changing the program's structure. Thus the version of the program described here should be taken as an illustration (realized on a computer) of the general principle of the program's structure.

3. Primitive Operators

3.1. Preliminary considerations. Primitive operators can be understood as individual "words" of the language describing distinguishing rules. The main part of this section represents the "dictionary" of this language. Every paragraph of this section contains a description of elementary operators on two principal levels.

The first level concerns the function of operators, which is already reflected in their names. The names of operators and the types of their input and output are listed on the next page. Even though at this level the descriptions of the primitive operators are not complete, they are sufficient for understanding the basic principles of the program's organization. Moreover, the external behavior of the program apparently does not depend on the concrete implementation of these operators. In other words, application of alternative variants of the algorithm of primitive operators satisfying the meaningful names given in the list would, it seems to us, produce results differing little from those obtained in real experiments with the program described here.

The second, more detailed level of description of operators concerns their structure. In this, emphasis is placed on the description of the function of individual operators rather than on their concrete machine realization. Their working algorithms are given only in case where such a description seems to be the clearest one. The explanations are illustrated by a number of examples.

One more problem raised in this chapter is the problem of the appropriateness of using a given operator in the program. It is not always possible to substantiate the use of each individual operator — such a question should be analyzed for the totality of all the primitive operators. If the chosen collection of operators allows the description of sufficiently varied distinguishing rules for a large number of interesting problems, we can say that the given language is appropriate for the solution of these problems. In the last paragraph of this section the reader can find several examples of "phrases" — complex operators built on the basis of primitive operators. This paragraph pursues a purely illustrative goal: it attempts to demonstrate, on one hand, the flexibility and versatility of the language being used, and, on the other hand, the limits of its capacity.

3.2. Drawing operators. ***3.2.1. Function.*** Drawing operators take a picture on the input and transform it into another picture. The results of work of three primitive drawing operators — *contour isolation*, *contour filling* and *convex hull filling* are shown in Fig. 1.

During the processing of the input picture the drawing operators can get information about the general character of the picture itself. This is achieved by comparing pictures on the input and the output of the operator. Their identity (the invariance of the input picture in relation to the given transformation) makes it possible, by using the operator *contour isolation,* to distinguish a "contour" picture from a "non-contour" one, and by using the operator *convex hull filling,* to distinguish a "convex" picture from a "concave" one. In this way, an additional output of drawing operators is obtained, namely, Boolean numbers. As a result of applying different drawing operators to collections of pictures, the output collections of Boolean numbers will define the subsets of "contour", "non-contour", "concave", "convex", etc., pictures.

Thus, a collection of pictures is supplied on the input of a drawing operator; on the output we get a collection of pictures and two (mutually complementary) collections of Boolean numbers, defining, respectively, two subsets of the input collection of pictures. One subset contains pictures that are invariant in relation to the work of the given operator; the other subset contains the non-invariant pictures.

List of elementary operators

A		
Area of picture	P_m	N_m
Length of lines	P_m	N_m
Coordinates of the center of gravity	P_m	$2N_m$
Slope of elongated picture	P_m	N_m
Main axes of picture	P_m	$2N_m$
Contour isolation (*C*)	P_m	P_m, $2B_m$
Contour filling (*F*)	P_m	P_m, $2B_m$
Convex hull filling (*T*)	P_m	P_m, $2B_m$
B		
Decision operator	B_1	—
Logical operator (*L*)	B_m	$4B_1$, $4B_2$, ..., $4B_{m-1}$
Threshold operator (*H*)	N_m, B_m	kB_m
Number of parts (*Q*)	B_m	N_1, N_2, ..., N_{m-1}
Comparison (*R*)	N_l, N_m, B_m	kB_m
Union (*U*)	P_m, B_m	P_1, P_2, ..., P_{m-1}
C		
Separation by connectedness (*S*)	P_m	P_{m+1}, B_{m+1}
Separation by borders (*J*)	P_m	P_{m+1}, B_{m+1}
Separation by branching nodes	P_m	P_{m+1}, B_{m+1}

Note. For each operator, types of input (first column to the right) and output (second column) collection of objects are indicated. *P*: collections of pictures; *N*: collections of numbers; *B*: collections of Boolean numbers. The bottom index is the number of level. $2N_m$ should be read as: “two collections of numbers of the m-th level”.

3.2.2. Structure. *Contour isolation (C)* Such elements of the picture are erased, which on the input picture have all the neighboring (in the sense 1) points black.

Contour filling (*F*). In the algorithm of contour filling an intermediary process F^* is used: the separation of the internal area F^*p of the initial picture p. The output of the operator contour filling is defined by the formula $Fp = p \cup F^*p$.

The separation of the internal area F^*p is done in the following way: 1) a negative $\bar{p}$ of the input picture is constructed; 2) on this negative remove all the connected in the sense 1 areas of the picture adjacent to the borders of the raster. The rest will be the internal area F^*p.

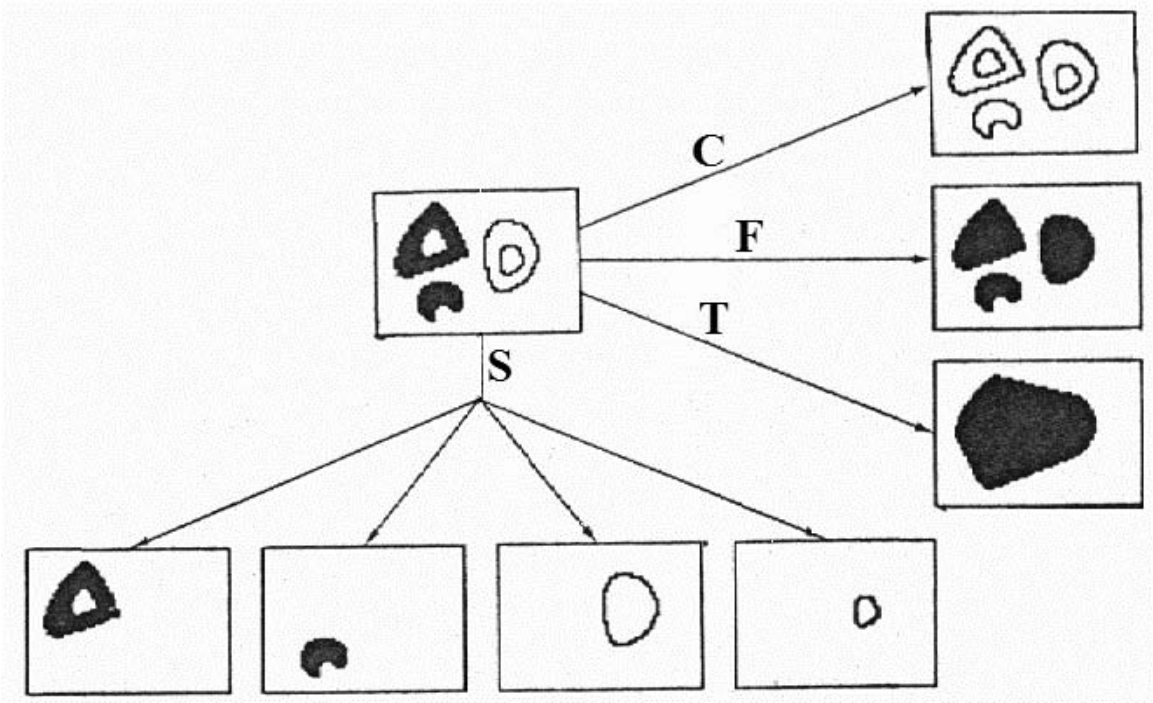


Figure 1. *Work of the drawing operators contour isolation C, contour filling F, convex hull filling T,* and *separation by connectedness S.*

The algorithm of erasing of connected areas coincides with the algorithm S^* of the separation of connected areas (see p. 29) in many details; consequently, we do not describe it here.

Filling of convex hull (*T*). First let us look at the auxiliary operations with the pictures that help build the operator convex hull: T_1 — constructing the negative of the upper part of the convex hull, and T_2 — constructing the negative of the lower part of the convex hull.

The construction of the negative of the upper part of convex hull is broken into several consecutive stages (their number depends on the character of the input picture p). A stage can turn out "successful" if it led to constructing a certain part of the negative of the convex hull, and "unsuccessful" in the contrary case. The construction of the picture T_1p at each stage is done consecutively by entire columns; an elementary step of a stage consists of adding a certain number of points of the next column to the already obtained picture T_1p.

Below, algorithm T_1 is given; its work is illustrated in Fig. 2.

Initially $T_1p = \phi$.

1. Moving from left to right we search at each column ξ_j of picture p until we see the first non-empty column; the columns on T_1p that correspond to the empty columns of picture p become black.
2. In this (first non-empty from left) column we find the highest excited point α.
3. Analogously, moving from the right border of the raster towards its left border, we look at the columns of picture p until we find the first non-empty column.
4. In this column we find the highest excited point ω.
5. Two points of the raster α and β are arguments of the main process (6) of algorithm T_1. Initially they are assigned the following values: $\alpha := \alpha, \beta := \omega$.
6. Moving from the column containing the point α to the right we consequently blacken on the picture T_1p all the points that lie above the straight line connecting the points α and β. This process can be aborted on two accounts:
   a) Picture T_1p intersects with p. We find the highest point of the column $p \cap T_1p$ and register it as a new value of β; then on picture T_1p we erase everything created on the last "unsuccessful" stage (after the previous assignment of values to the arguments α and β), and repeat the process 6 (with the new value of β).
   b) The straight line connecting α and β went outside the borders of the raster. If in this case $\beta = \omega$ then the construction of the upper part of the negative of the convex hull T_1p is complete. If $\beta \neq \omega$, we assign new values to the arguments of process (6): $\alpha := \beta$ and $\beta := \omega$, and the process (6) is repeated. Analogously for the negative of the lower part of the convex hull T_2p.

Let us denote p_1 the picture

$$p_1 = \overline{(T_1 p \cup T_2 p)}$$

Then the algorithm of filling the convex hull shall be

$$T_p = \begin{cases} p, & \text{if } p_1 \setminus C_p \subseteq p \\ p_1, & \text{otherwise} \end{cases}$$

Boolean outputs of drawing operators. For every drawing operator D the two collections of Boolean numbers $b_1^i \in B_{D_1}$ and $b_2^i \in B_{D_2}$ that appear as its output are defined in the following way

$$b_1^i = \begin{cases} 1, & \text{if } Dp^i = p^i \\ 0, & \text{if } Dp^i \neq p^i \end{cases}$$

$$b_2^i = \overline{b}_1^i$$

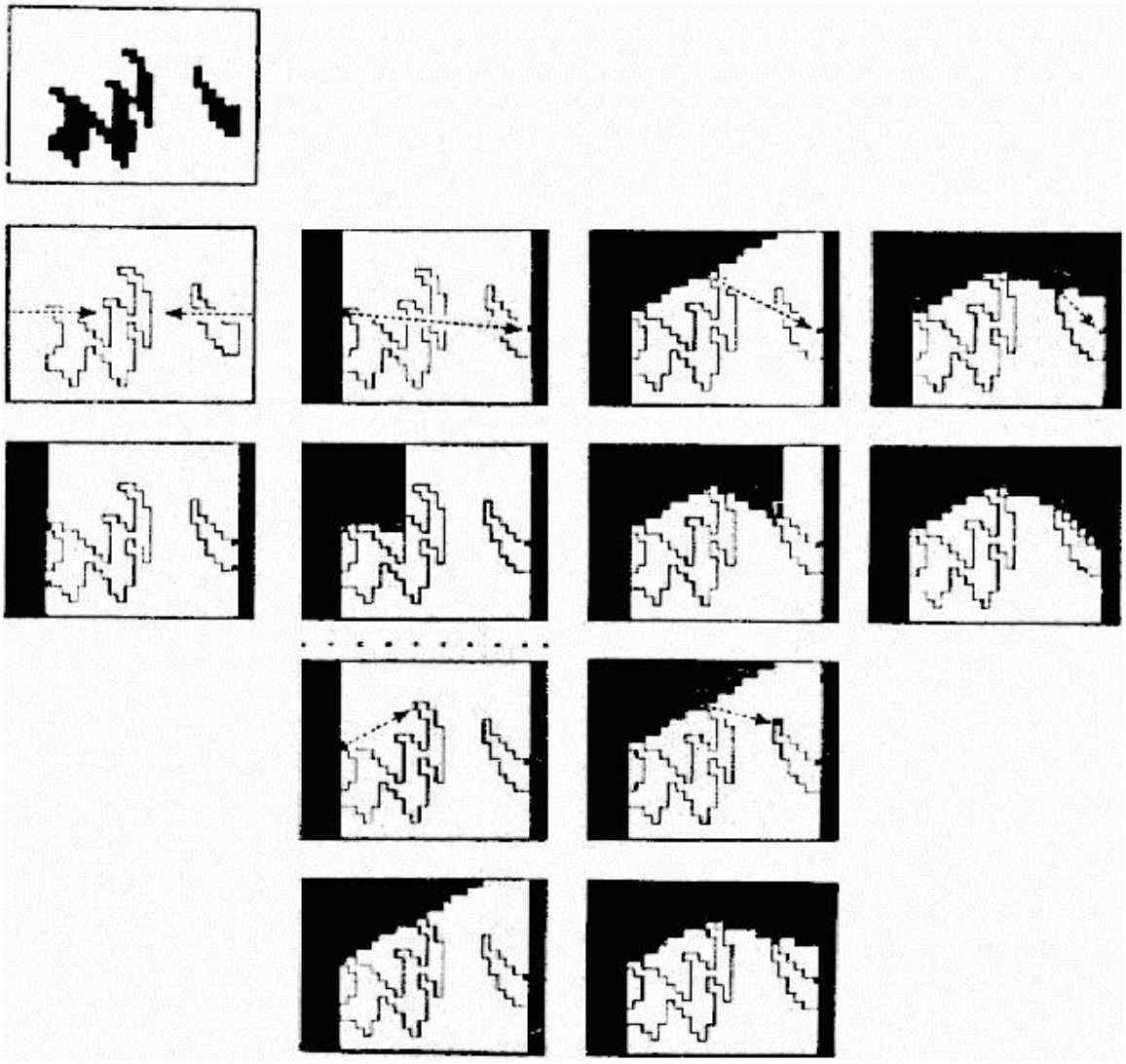


Figure 2. *An example of constructing the negative of the upper part of convex hull.* On the upper left is the input picture. The four columns of pictures under it represent the sequence of intermediate results of building the output picture. The pairs of consecutive pictures in these columns represent the beginning and the end of one stage of the work of this algorithm. Each of the columns ends by a "successful" stage.

3.3. Measuring operators. ***3.3.1. Objective.*** The measuring operators put certain numbers (results of measurement) in correspondence with each input picture. The usefulness of such operators for building distinguishing rules is obvious. Indeed, in these problems classes can differ by size or by location of elements in a picture (see Problems #22 and #36 in the Appendix), by the spatial orientation of elements (for example in Problem 5), etc. Thus it is helpful to measure the corresponding parameters of elements in order to use the measurement results for classification.

In the program the following measuring rules are represented (see the list): *coordinates of the center of gravity, area of the figure, length of lines*. (The last operator should be applied only to pictures consisting of lines. Anticipating the following material, let us notice that the operator *length* is applied only after the application of the drawing

operator *contour isolation*.) In addition, some parameters are measured that characterize the length and width of a figure — the values of its small and big axes. These parameters we will call *values of the main axes* of a figure. If the picture has considerable eccentricity then the *slope* of the longitudinal (big) axis is measured. The measuring of values of both small and big axes of a figure, as well as of the slope of its longitudinal axis are all combined into a single operator.

3.3.2. Structure. The *area* is defined according to the number of black points

$s = \sum_{i,j} \alpha_{ij}$.

Coordinates of the center of gravity: the outputs of this operator are as follows:

$$n_x = X_p = \begin{cases} \frac{1}{s}\sum_{i,j} j\alpha_{ij} - 1 & for\ s \neq 0, \quad \frac{1}{s}\sum_{i,j} j\alpha_{ij} \leq 63, \\ 62 & for\ s \neq 0, \quad \frac{1}{s}\sum_{i,j} j\alpha_{ij} = 64, \\ f & for\ s = 0. \end{cases}$$

$$n_y = Y_p = \begin{cases} \frac{1}{s}\sum_{i,j} i\alpha_{ij} - 1 & for\ s \neq 0, \\ f & for\ s = 0. \end{cases}$$

Length of lines. Let us see the structure of this operator in more detail. The length of lines on a contour picture could also be to a certain extent calculated according to the number of the black points on this picture. However, such an appraisal is unsatisfactory on the following accounts: 1) the contour of double thickness (Fig 3, b) contains twice as many black points as does the single contour of the same length (Fig. 3, a); 2) the quantity of black points in a rectangular segment depends on the slope and, all other conditions being equal, can become $\sqrt{2}$ times as big or as small (Fig. 4).

The first difficulty can be avoided by making the double lines twice as thin.

This procedure consists in consecutive erasing of some individual black points from the initial picture. Whether a point is to be erased is defined solely by the configuration of excitement of its neighbors. The character of excitement of all the other points of the raster plays no role here. The four types of elements to be erased (differing by the 90° rotation of its configuration) are shown in Fig. 5. The thinning of lines is performed in entire columns — the elementary stage (1) consists of erasing a certain set of points in current column in a given picture. Step 2 of the procedure defines the order in which the columns are processed. A procedure for the thinning of lines is shown on Fig. 3, b, c, d.

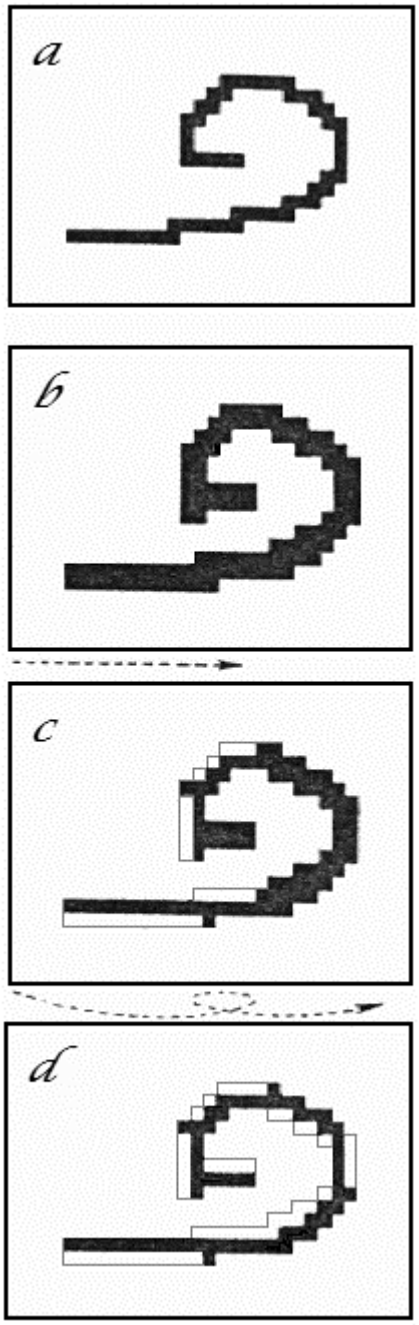


Figure 3. *Line thinning*

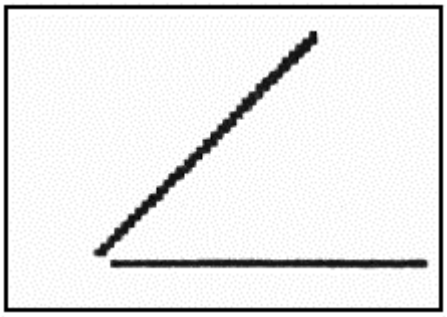


Figure 4. *Equivalent straight line -segments.*

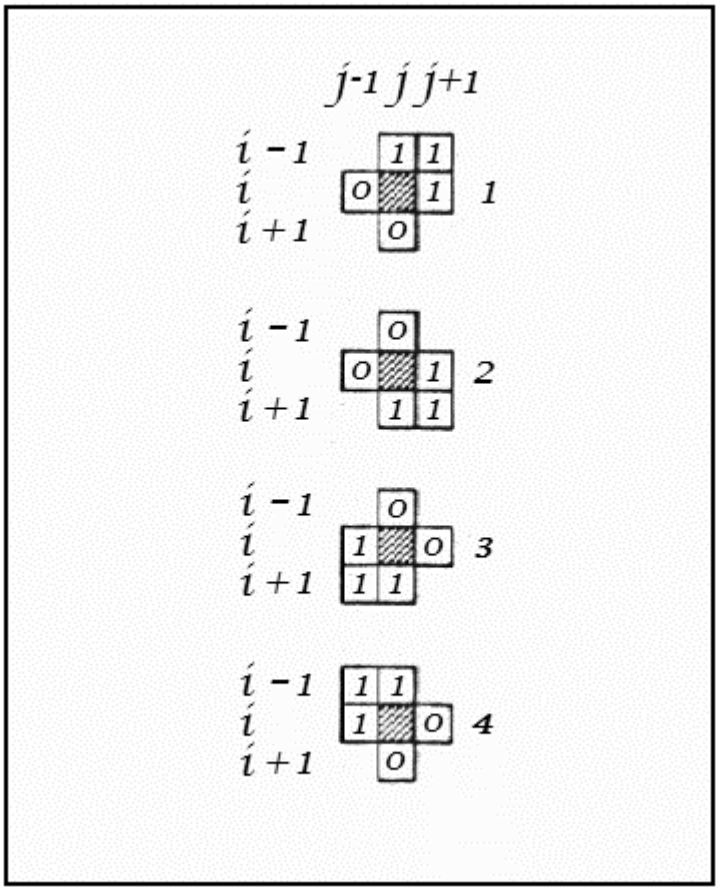


Figure 5. *The four types of erasable elements in the procedure of line thinning.*

Procedure (*V*) starts from column $j = 1$.

1. In this column, the erasable elements of all four types are consecutively identified and erased (Fig. 5):
   a) First we find all the erasable elements of the 1st type; if there are none, we pass to point *b*; if we find such elements, we erase them and repeat the process starting from point *a* of the same column;

b) On the resulting picture we find all the erasable elements of the 2nd type; if there are none, we pass to point *c*; if we find such elements, we erase them and repeat the process starting from point *b* of the same column;

c) Analogously we erase the elements of the 3rd type;

d) On the resulting picture we find all the erasable elements of the 4th type; if there are none, the application of process 1 to the given column ends; if we find such elements, we erase them and repeat the process starting from point *c* of the same column.

2. After that the following situations are possible:

a) At the previous step no erasable elements of the 3rd or 4th types were found. If the processed column was not the last one ($j \neq 64$), then process 1 is applied to the following, j+1-th column; in the contrary case the line-thinning procedure is complete.

b) At the previous step some elements of the 3rd or 4th types were erased. In this case we come back one step and apply the process 1 to the previous j–1-th column.

After line thinning the number of black points s and the number of edge-points s' is calculated. A point is called an edge-point if it is black and there exists at least one pair of neighboring (in the sense 2) black points ($\alpha_{i_1 j_1}$, $\alpha_{i_2 j_2}$) such that they are neighboring (in the sense 1) in relation to each other. Examples: on the horizontal segment of Fig 4 there are no edge-points; on the sloping segment all the points except the ends are edge- points.

Finally, the length of lines is defined by the formula:

$$l = s - (1 - \sqrt{2}/2)\, s'.$$

Main axes of a figure. When working out an algorithm of measuring the length of the figure's axes we have used the fact that the principal moments of inertia of an elliptical figure are proportional to the squares of the big and small axes of the ellipse. It allows us to reduce the measurement of the length of the width of a figure to purely computational operations: the calculating of the principal moments of inertia, and the measurement of the slope of the main axis of the figure, to calculating the angle between the main axes of inertia of the figure and the axes of coordinates of the raster. For this it is necessary to first calculate the components of the inertia tensor of the figure:

$$I_{11} = \sum_{i,j} i^2 \alpha_{ij} - \frac{1}{s}\left(\sum_{i,j} i\alpha_{ij}\right)^2,$$

$$I_{22} = \sum_{i,j} j^2 \alpha_{ij} - \frac{1}{s}\left(\sum_{i,j} j\alpha_{ij}\right)^2,$$

$$I_{12} = -\sum_{i,j} ij\alpha_{ij} + \frac{1}{s}\sum_{i,j} i\alpha_{ij} \sum_{i,j} j\alpha_{ij}.$$

The values of the main axes of the figure are calculated by the following formulae:

$$a_1^2 = \frac{I_{11} + I_{22} + \sqrt{(I_{11} - I_{22})^2 + 4I_{12}^2}}{2s},$$

$$a_2^2 = \frac{I_{11} + I_{22} - \sqrt{(I_{11} - I_{22})^2 + 4I_{12}^2}}{2s}.$$

If the difference between the values of the main axes a_1 and a_2 is less than 10%, i.e.,

$$2\left|\frac{a_1 - a_2}{a_1 + a_2}\right| < 0.1,$$

then the measured figure is considered "round" and has no slope ($n_\varphi = f$). If the above condition is not fulfilled, the relation $tan(2\varphi) = 2I_{12} / (I_{11} - I_{22})$ is used to find the slope.

The values of the main axes and the slope are calculated according to the formulae given above if $s \neq 0$. If $s = 0$, we suppose that $a_1 = a_2 = 0$, $n_\varphi = f$.

Logarithmic scale. Area of figure, length of lines and values of axes are coded in the logarithmic scale. For this, a general subprogram of taking logarithms is used in the corresponding measuring operators. The arguments of the subprogram — s, l^2, a_1^2, or a_2^2 — are transformed into six-digit binary codes which, precisely, are the numerical output of the said measuring operators.

Since the outputs of the logarithm-taking program can assume only 63 (integral) values, the choice of the base of logarithms plays an important role. On one hand, the approximation as a result of rounding off should not be too big; on the other hand, these 63 values should encompass the entire range of the possible values of these arguments. In the described subprogram this base was chosen equal to $\sqrt[4]{2}$. As a result the area was coded with a 20% precision (in other words, a twofold change of area corresponded to a change of logarithm by four gradations). The coding precision of length of lines and of axes values turns our to be twice as large, since in these cases the arguments of the logarithm-subprogram are squares: l^2, a_1^2, a_2^2.

The slope of the horizontal axes of the figure is coded very approximately — only 8 gradations of non-oriented slope are distinguished (Fig. 6): $n_\varphi = 0, 1, \ldots, 7, f$.

3.4. Transformation of numbers into Boolean numbers

3.4.1. On the appropriateness of using threshold operators.

In order to be able to use the measuring operators for building distinguishing rules, we need to be able to transform the output objects of these operators (numbers) into Boolean numbers. Indeed, a distinguishing rule is a complex operator that is applied to pictures and gives as its output Boolean numbers.

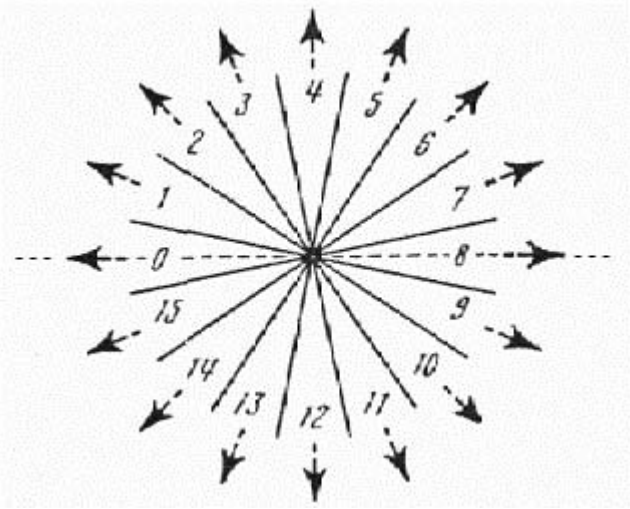


Figure 6. *Eight gradations of slope and sixteen gradations of direction.*

The common procedure of transformation of numbers into Boolean numbers consists of inputting predicates, for instance, the type of comparing with a threshold (or several thresholds) and re-coding the numbers into Boolean zeros and ones according to the area (related to threshold(s)) into which these numbers fall.

It is useful to include such threshold operators into the list of primitive operators for building distinguishing rules. They should be included as a sequence-measuring operator — threshold operator. The application of "threshold operators" to the results of work of different measuring operators (length, area, coordinates of center of gravity, etc.) allows us to describe the pictures in terms that are equivalent to such common "human" terms as "long", "short", "big", "medium", "small", "left," and so on.

Of course it is not always the case that distinguishing rules boil down to measurements with the following cutting off along at the threshold. In the simplest problems such a pair of operators can, indeed, directly define the corresponding distinguishing rule. Examples of this are problems in which classes are distinguished by the area of figures in the pictures ("big"–"small"), length of lines ("long"–"short", see Problem #22), slope of figures ("vertical"–"horizontal", see Problem #5), etc. However, in more complicated problems such concepts as "big" and "small" can only be a part of the description of the distinguishing rule, defining the subset of pictures in which big (or, correspondingly, small) figures are depicted (see Problem #42).

Speaking about threshold operators, it is necessary to notice that the very expression "use of operators" is ambiguous. Primitive operators are used, on one hand, as words of the language that describes distinguishing rules and, on the other hand, for searching for these rules in the process of problem solving. As it has been shown, the threshold operators are necessary for the description of distinguishing rules. Here we shall analyze which threshold operators (or, more precisely, which values of thresholds and for which parameters) can be appropriately used in the process of executing combinations of primitive operators in the search for distinguishing rules.

3.4.2. Possible ways of choosing thresholds in the search for distinguishing rules. One of the ways of establishing thresholds is evident. If the maximal value of a certain

parameter for pictures of one class is smaller than its minimal value for the pictures of the other class, it is enough to establish a threshold between these two values. Precisely this threshold will define the distinguishing rule in this case. Such a way of establishing thresholds could be successfully used in solving certain problems. Unfortunately, it is useful only in those simplest cases when distinguishing parameters according to the threshold is the final step in the search for the distinguishing rule.

Another way consists of trying all the variants of thresholds for each parameter hoping that some of the collections of Boolean numbers (obtained as a result of separating collections of numbers according to thresholds) would be useful for future construction of the distinguishing rule. The search tree in this case would have a large number of branches, since each result of measurements can be converted to collections of Boolean numbers in many ways (setting different threshold values). In case of such an unlimited use of threshold operators it can be guaranteed that the threshold operator necessary for the distinguishing rule will be found. Regrettably, this method would also produce many distinguishing rules totally implausible from the human point of view.

Distinguishing rules satisfying only the given material but incorrectly distinguishing the entire set of objects (which will be found in the given problem upon examination) will be called *superstitions* [5]. The fact that there is a large number of superstitions among such distinguishing rules that can be constructed by execution of all the possible variants of thresholds forces us to give up this method. Having constructed all the possible distinguishing rules, we cannot get rid of all those that seem unnatural from the human point of view. For this we lack the formal criteria to judge whether an already constructed rule corresponds to the given problem. Moreover, the aim of the present work is to find a constructive method of building satisfying distinguishing rules. This is why it is necessary to introduce an additional restriction on searching by cutting those branches of search that lead to superstitions.

Thus, a certain criterion of choosing the threshold values is necessary for the application of threshold operators in search of distinguishing rules. Such a criterion must, on one hand, choose in each problem, in each collection of numbers only those threshold values that can lead to the solution, while discarding the paths leading to superstitions. On the other hand, the criterion must be local, that is, it should allow discarding completely the following branches of search without analyzing the possible consequences, using only the data contained in the input collection of numbers.

3.4.3. Local criterion of choice of threshold values. This criterion was proposed by M. Bongard under the name *"breaking up into lots"* [5]. In continuation, we will give arguments in support of this criterion and describe psychological experiments supporting the idea that a similar criterion is applied by people in the solving of geometric problems.

The criterion is based on the following idea. Let it be that in a certain problem (but not in the sum total of all problems being solved) all the numbers in a certain collection of numbers can have only two values. Then it would be sufficient to introduce a single threshold dividing these values. All the other threshold values would lead only to a meaningless increase of the search.

As a rule, numbers in the collection take many values. Sometimes they may be grouped around several values (forming "lots"). It seems natural to apply the above-mentioned

procedure also in this case, establishing the threshold values among these lots and thus similarly coding the numbers that get into one lot. In this way it is possible to choose the threshold values for such particular cases when numbers in a collection (or numbers in the sum total of all numbers) take only a small number of values or are grouped around a small number of values.

What to do, however, in case of a large number of values that cannot be broken up into lots? An essential characteristic of the proposed method is that a collection of numbers is coded in Boolean numbers only in case it can be broken up into lots. Otherwise the coding is not done. This can be justified by the fact that, as it will be shown below, the problems in which the important parameter for building a distinguishing rule cannot be broken up into lots are difficult for people or cannot be solved by people. This is why the program that uses this criterion will not be able to solve precisely those problems that are difficult for people. Moreover, a collection of numbers that is not divisible into lots can nonetheless be used in building a distinguishing rule, in case a subset of the given collection or a certain combination of parameters (non divisible into lots) can be broken up. As examples we can cite problems #30 and #34, in which the area of figures is not divisible into lots, but makes part of the description of the corresponding distinguishing rules. In §3.4.5 and §3.6 we describe the operations necessary for this.

Of course, the above considerations cannot be seen as a sufficient argument for this criterion. Its appropriateness is understood intuitively in each concrete case of constructing drawing and measuring operators. Thus we did not conduct special psychological experiments showing that such operations can be used by people in solving problems. On the other hand, the proposed criterion of threshold selection by coding numbers with Boolean numbers is not evident and, therefore, required such experiments.

On M. Bongard's suggestion, L. Dunayevsky conducted a series of experiments [5]. The subjects were presented with problems consisting of 24 pictures separated into two classes. The figures on the pictures differed from one another by six parameters. Two of those parameters (different in different problems) were part of the description of the classification principle. The experiments showed that if one of the two essential parameters cannot be broken up into lots, the average time of solving the problem increases almost twofold (as compared with the case when both parameters can be broken up. If none of the two parameters can be broken up, the solution time increases even more, or the subjects simply refuse to solve the problem. It must be noted that the parameters were standardized and their number was restricted; this was done in order to obtain uniform quantitative results. Such restrictions, as well as a comparatively large collection of training pictures used in the experiments, make it easier for people to find the solution in cases where a threshold based on a non-divisible parameter must be chosen. Usually, even one non-divisible parameter is enough to stop people from solving the problem.

In sum, the experiments have shown that people solve easier such problems in which the essential parameters can be broken up into lots. Yet the true experimental verification of the fact that for a large number of geometric problems the criterion of breaking up into lots is really effective for choosing thresholds (in the sense of cutting off "hopeless" branches of the search) can be obtained only while training a system that uses the procedure of breaking up into lots as one of its internal blocks. For this it is necessary to

first formalize the criterion of breaking up into lots — to find a precise algorithm allowing us to define, from the input collection of numbers, the thresholds breaking these numbers into lots.

3.4.4. Algorithm of breaking up into lots. The algorithm consists of several dissimilar parts and contains much branching. Some parts are evident, while others have to be substantiated. As a result, in the description of the individual parts of the algorithm we had to answer not only the question "how" they are made, but "why" these parts are made precisely this way. This is why we opted for describing the main ideas first (not in the precise order in which they appear in the algorithm). Then we will briefly characterize the general structure of the algorithm and its modifications.

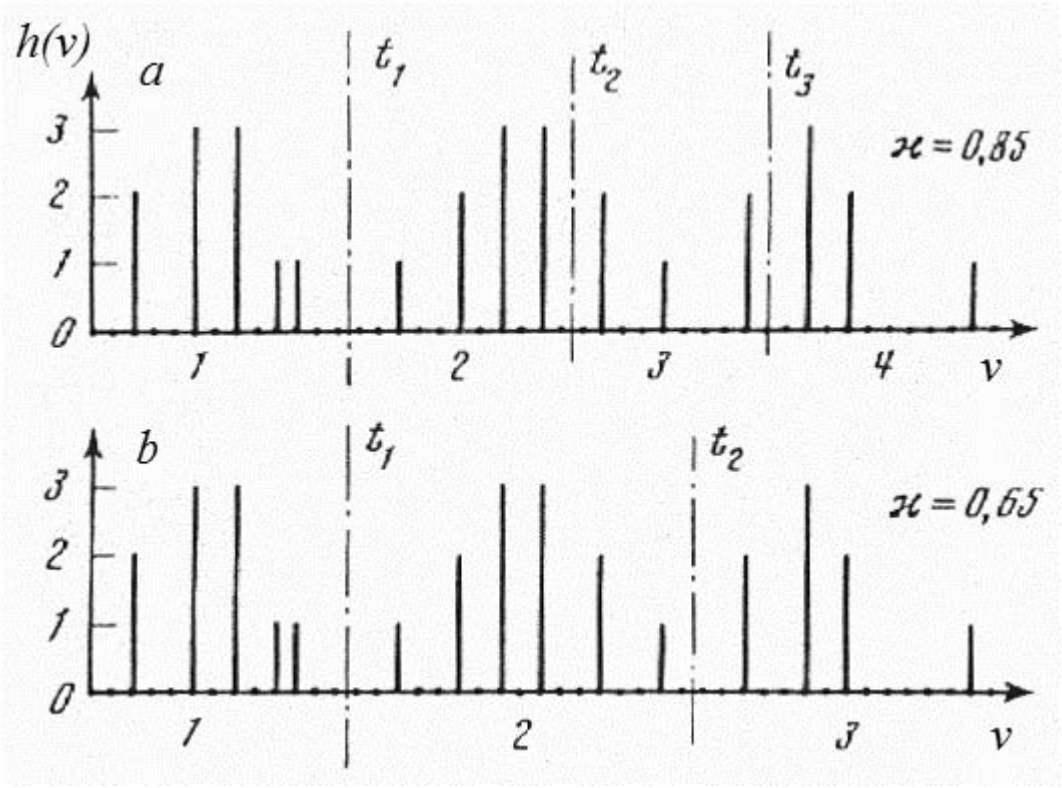


Figure 7. *Two variants of breaking up the bar chart.*

The process of finding the distinguishing thresholds is built in the following way. First a bar chart is constructed $h(\nu)$, $\nu = 0, 1, \ldots, 62$. It shows how many numbers of the given value ν are contained in the input collection. From now on the algorithm works only with this bar chart.

Let us suppose that we have already introduced some thresholds $t_1, t_2, \ldots, t_{k-1}$ that divide the chart into k areas (Fig. 7). For each of these divisions, let us introduce a numerical value κ, the index of compactness. Let q be the volume of the sum total of numbers,

$q = \sum_{\nu=0}^{62} h(\nu)$; σ^2 is the *variance*:

$$\sigma^2 = \frac{1}{q-1}\sum_{\nu=0}^{62}\nu^2 h(\nu) - \left[\frac{1}{q}\sum_{\nu=0}^{62}\nu\, h(\nu)\right]^2;$$

q_i is the volume of the i-th area $\left(\sum_{i=1}^{k} q_i = q\right)$, and σ_i^2 is the variance of numbers that turned out to be in this area. Then the index of compactness for the given system of thresholds is calculated by the formula

$$\kappa = k^2 \frac{\sum_{i=1}^{k}(q_i - 1)\sigma_i^2}{(q-k)\sigma^2}.$$

With such a system of threshold-evaluation it is feasible to choose the best among all possible divisions of the bar chart into areas: the division with the minimal value of compactness index κ_{min}. If κ_{min} is smaller than a certain preset value of κ^* then the problem is solved: the best system of thresholds is found breaking up the totality of numbers into lots. If $\kappa_{min} \geq \kappa^*$ then the given group of numbers cannot be broken into lots.

The value of κ^* was found experimentally. In the first variants of the program κ^* was given the value of ⅜. Then we found out that this value can be reduced to ¼. These cases were not much different from each other. In the majority of cases, in the first cases the breaking up into lots occurred more frequently. Since at each such division the program prints out the value of κ_{min}, the records showed that all the relevant (from the point of view of the distinguishing rule) break-ups occurred in all problems with rather small value of κ_{min}. It is difficult to say anything more precise about this experiment, since the program's behavior greatly depends on the kind of problem given. Apparently, the changes in the program, brought about by the change of κ^* were too insignificant to be able to draw general conclusions. All the experiments described in §5.5. were conducted with $\kappa^* = ¼$.

In addition to the numerical estimate of divisions, the following restrictions for the possible divisions of groups of numbers were used:

1. $q \geq 8$ — small groups of numbers cannot be broken into lots;
2. $q_i > q/8$ ($i = 1, 2, \ldots, k$) — the lots themselves should not be too small;
3. $k \leq 4$ — the groups of numbers can be broken only into 2, 3, or 4 lots.

While working out the algorithm of breaking up into lots we paid special attention to the speed of work of the program. As the experiments have shown, the number of possible trial break-ups (different subsets of numbers) in some problems can reach several thousands. Since the calculation of the break-up estimate κ is a rather lengthy procedure, it is not feasible to do the full execution of all the possible divisions of the bar chart into different number of areas, even while taking into account the restrictions 1–3.

At the same time there exist situations (for example, the case of a uniform distribution of numbers in a group), when it is possible to say without the execution (judging on other

criteria) that the bar chart cannot be broken up into lots. Analogously, it is possible to find the local criteria that would allow us to estimate for each area of the bar chart whether the distinguishing threshold is possible there. To speed up the work of the algorithm of breaking up into lots, we have used some *a priori* "quick" criteria of the choice of thresholds that allow us to substantially reduce the search.

Let us number all the values ν_j, $j = 0, 1, 2, ..., J$, such that $h(\nu_j) \neq 0$ in the following order: $\nu_0 < \nu_1 < \ldots < \nu_J$. Now let us analyze the intervals $(\nu_{j-1}, \nu_j,)$, $j = 1, 2, \ldots, J$.

The reduction of execution consists in that from the sum total of all such intervals of the bar chart (ν_0, ν_1), (ν_1, ν_2), …, (ν_{j-1}, ν_j), a certain number of intervals–candidates is selected, inside of which it is permitted to introduce distinguishing thresholds. Then the program analyzes all the possible combinations of only those intervals–candidates that define the division of the bar chart into areas. For each of such divisions, according to the above described scheme, the estimate κ is calculated and the division with the minimal value of κ is chosen.

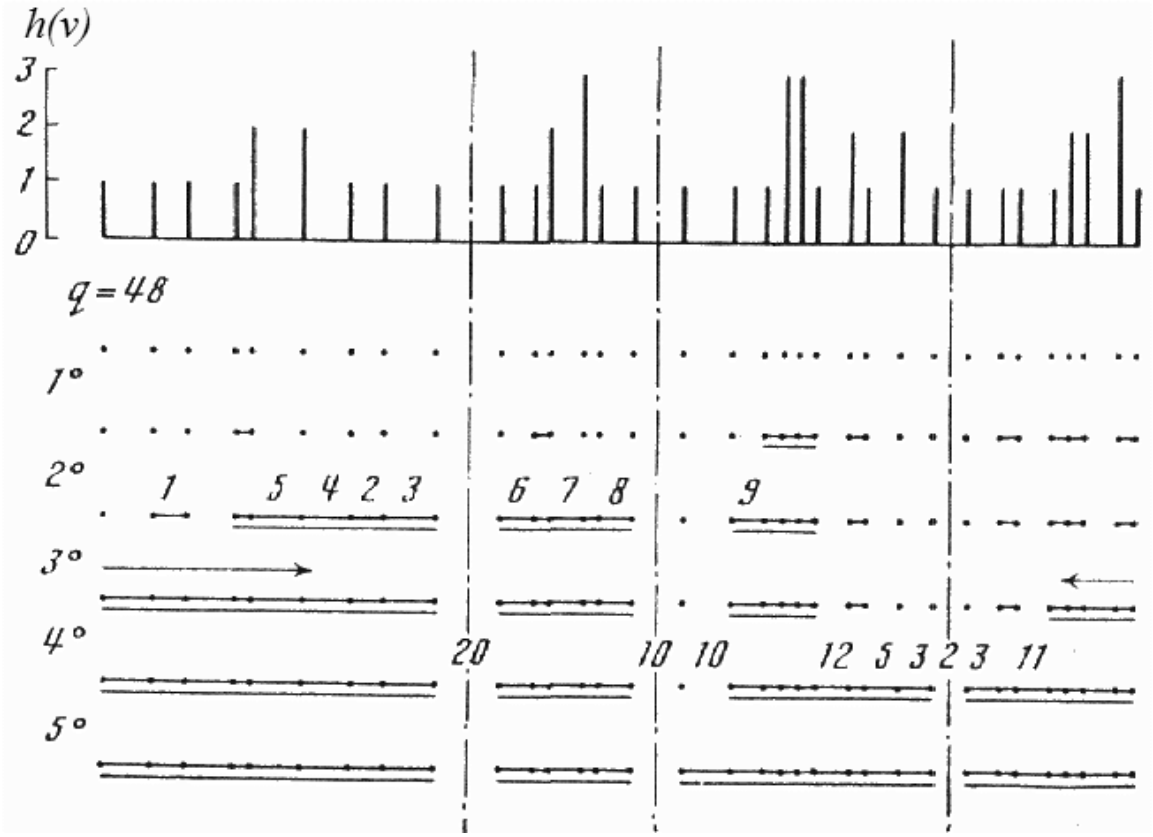


Figure 8. *Procedure of preliminary selection of intervals inside of which distinguishing thresholds can be established.* On top, the initial bar chart $h(\nu)$. Above, fine consecutive steps of eliminating intervals between the points $h(\nu) \neq 0$. The big areas are underlined. 1: elimination of small intervals; 2: comparison of pairs of intervals; the numbers under the intervals correspond to the order in which the intervals are eliminated; 3: "gluing together" the "tails" (ends) of the bar chart; 4: elimination of equal intervals (in sequences between large areas). The summary volume of the two areas broken up by each interval is written above the corresponding intervals.

The selection of intervals–candidates occurs in several consecutive steps. At each step certain intervals are eliminated from the sum total of intervals. The intervals remaining as a result of the work of this algorithm are considered intervals–candidates. Fig. 8 illustrates the work of the algorithm of the selection of intervals–candidates.

1. Cross out all the small intervals. From the sum total of all the intervals keep only those that satisfy the condition that $\nu_j - \nu_{j-1} > 3\bar{\rho}/2$, where $\bar{\rho} = (\nu_j - \nu_0)/(q-1)$.

2. Comparison of pairs of intervals.

Intervals that have not been eliminated divide the bar chart into several areas. The volume of each of these areas $\Sigma h(\nu)$ is calculated (addition is made over all ν in the given area). We will distinguish big areas (whose volume is > q/8) and small ones (whose volume is ≤ q/8). We analyze only such pairs of not-eliminated consecutive intervals that are separated by small areas.

We do consecutive execution (starting from intervals with small numbers) of such pairs until the first pair of non-equal intervals is found. If such a pair exists, we cross out the smaller interval of the pair and repeat procedure 2 from the start. If among the not-eliminated intervals such a pair of consecutive non-equal intervals separated by small areas is not found, we pass to the next step.

3. Restriction 2 on page 20 defines the upper and lower limits for the first and last of the intervals–candidates, respectively: each one of them must separate from the bar chart by more than ⅛ from the total volume of the sum total q. We eliminate all the intervals (ν_{j-1}, ν_j,), j = 1, 2, …, J, for which

$$\sum_{l=0}^{j-1} h(\nu_l) \le q/8 \text{ or } \sum_{l=j}^{J} h(\nu_l) \le q/8$$

As a result of the procedures 2 and 3 the bar chart is broken up into several big areas, among which chains of small areas may appear, separated from one another by equal (inside the limits of the entire chain) intervals (Fig. 8).

4. We analyze one by one all these chains that separate big areas. For each interval of a chain we calculate the summary volume of the two areas it separates. We leave the intervals for which this summary volume is minimal, and eliminate all the other intervals.

(On the sample bar chart on Fig. 8, at the beginning of step 4, there exist three separating chains of intervals. The first “chain” contains only one interval; it is not eliminated. The second one contains two intervals. The summary volumes of the contiguous areas (numbers above these intervals) are equal; thus those intervals are not eliminated. The third chain contains six intervals, only one of which remains.)

5. If after this there still remain chains of intervals divided by small areas in the bar chart, then we keep the interval with the smallest number in each chain. All the other intervals are eliminated.

Thus, the principal steps of the algorithm of breaking up into lots are:

A. Constructing a bar chart

B. Search for the intervals-candidates.

C. Execution of the possible combinations of the intervals–candidates — the choice of the best break-up.

At each of those steps it can turn out that the given group of numbers cannot be broken up into lots. This happens if

a) The group is too small ($q < 8$), or there is at least one "non-measurable" value in this group,

b) At some stage of preliminary choice of thresholds it turns out that there are no intervals-candidates,

c) $\kappa_{min} \geq \kappa^*$.

The case in which breaking up into lots is not possible is most frequently found already at the first stages of the algorithm's work. This allows us to save a significant amount of time. In the most complicated case ("poorly ordered" large group of numbers) the work of the entire algorithm takes up about 0.2 sec.

Modifications of algorithm defined by dimension.

Depending on the dimension of the input group of numbers, various modifications of the algorithm of breaking up into lots can be used. The following three modifications of the algorithm correspond to the three dimensions:

1) "Natural numbers" — outputs of the operator *number of parts* (page 36)

2) "Continuous numbers" — outputs of all measuring operators, except the operator *slope* of the figure.

3) Slope and orientation.

The described algorithm of breaking up into lots corresponds to the first modification and is applied to the group of numbers that by their nature are natural numbers. For instance, the number of figures in a picture can be expressed only by integers. On the other hand, the "continuous" outputs of the measuring operators mentioned above take integral values $\nu = 0, 1, 2, ..., 62$ as a result of rounding up. If, for instance, in the second group we find only numbers with neighboring values ν_0 and ν_0+1 and those are natural numbers, the group will break up into two lots ($\sigma_i^2 = 0$; $i = 1, 2$, and, consequently, $\kappa_{min}=0$). On the contrary, if they are continuous numbers (for instance, the result of work of the measuring operator area on a certain group of pictures), it does not mean that the areas of figures on the corresponding pictures take only these two values. In reality areas can be "spread" inside the intervals $[\nu_0, \nu_0+1)$ and $[\nu_0+1, \nu_0+2)$ and should not be broken up into lots. Hence the ways to calculate the continuity of the groups of numbers:

a) Distinguishing thresholds can be introduced only in such places where there is enough space between the areas;

b) The discrete bar chart should be replaced by its continuous variant consisting of rectangles of unitary width (Fig. 9), and all the sums in ν (while calculating the variance in the estimate of κ) should be replaced by the corresponding integrals.

Thus, the second modification of the algorithm of breaking up into lots consists of the following:

a) Modifications of step 1 of the procedure of choosing the intervals–candidates — if $3\overline{\rho}/2 \geq 1$, then the previous variant works; if $3\overline{\rho}/2 < 1$, then all the variants (ν_{j-1}, ν_j,) for which at the same time $\nu_j - \nu_{j-1} = 1$, $h(\nu_{j-1}) > q/8$, $h(\nu_j) > q/8$, should be crossed out;

b) The variances in case of continuous bar chart $\tilde{\sigma}_i^2, \tilde{\sigma}^2$ can be easily expressed through the variances of discrete bar chart (p. 19)

$$\tilde{\sigma}^2 = \sigma^2 + 1/12,\ \tilde{\sigma}_i^2 = \sigma_i^2 + 1/12.$$

As a result, the estimate κ in this modification of the algorithm is calculated by the formula:

$$\kappa = k^2 \frac{\sum_{i=1}^{k} (q_i - 1)\sigma_i^2 + (q-k)\,1/12}{(q-k)\left(\sigma^2 + 1/12\right)}.$$

Groups of numbers with a dimension of slope, in addition of being continuous, have yet one more particular characteristic: the slope is defined with a precision of 180° period, which corresponds to the 8 gradations of slope $\nu' = \nu \pm 8n$, where $n = 0, 1, 2, \ldots$ Since the bar chart of such a group of numbers $h(\nu')$ should be a periodic function, the algorithm of breaking up into lots is not applicable here in the form described above. The changes of algorithm that correspond to the third modification boil down mostly to the changes of step A, the construction of the bar chart. From the periodic bar chart $h(\nu')$ we isolate a fragment of the length of period of minimal variance σ^2 (Fig. 10). Then the second modification of the algorithm described above is applied to this fragment of the bar chart.

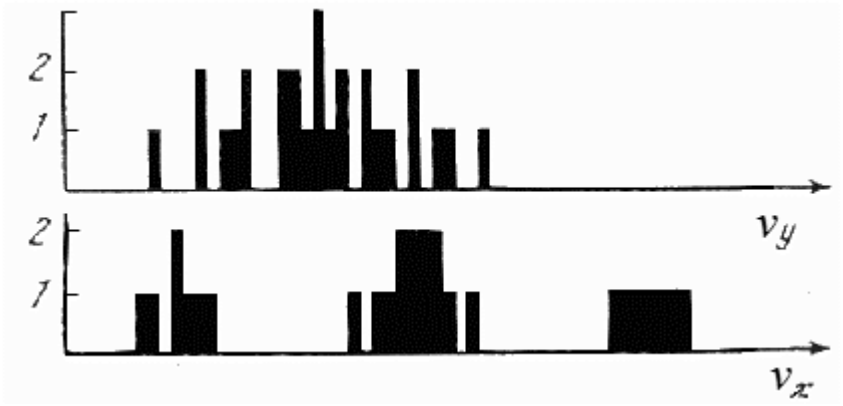


Figure 9. *Bar charts of the coordinates of the center of gravity of individual squares (parts of pictures) of Problem #27.* The y-coordinates do not break up into lots; the x-coordinates break up into 3 lots.

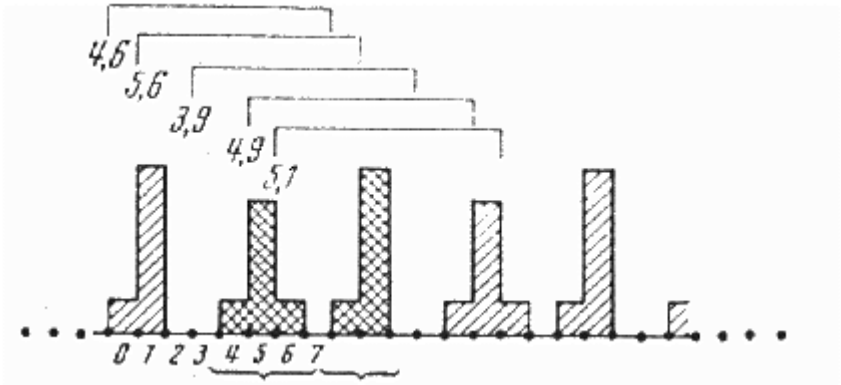


Figure 10. *Breaking up into lots of the slop of the figure's longitudinal axes in Problem #5.* The brackets above the "periodic bar chart" mark its different period — long fragment. The fragment with the minimal variance is chosen (the values of variances are shown near the brackets). The braces on the bottom mark the two lots into which the slope was broken up in the given problem. Analogously, for the group of numbers with the dimension *orientation* (see §3.6) we isolate a fragment of the periodic bar chart with a period of 360° that corresponds to the 16 gradations of orientation (direction).

3.4.5. Threshold operator. Function and structure. A threshold operator transforms collections of numbers into collections of Boolean numbers. Let us look at an example. At the input of this operator we have a collection of numbers characterizing a certain set of pictures, for example, the results of work of the measuring operator area on this set of pictures. If it turns out that, according to this parameter, the figures on the given pictures evidently break up into big ones and small ones without any intermediate area values, this operator will calculate (with the help of the algorithm of breaking up into lots) the threshold area value. Then it will compare each area value to this threshold and in this way each picture will be put into correspondence with Boolean numbers which can be interpreted as truth values of the following statements: 1) "in the picture a small figure is represented" and 2) "in the picture a big figure is represented".

It can happen that a certain collection of numbers can be broken up into three or four lots, instead of only two as in the previous example. Then the algorithm of breaking up into lots will calculate several distinguishing thresholds and the input collection of numbers will be transformed into k collections of Boolean numbers (according to the number of lots). In each of these collections of Boolean numbers the ones will correspond only to those numbers that fell into the given lot.

It can happen that the collection of numbers will be divisible into lots. In this case it is useful to try to break up into lots certain subsets of the numbers of this collection. As it was already said in Section 2, a subset of numbers N_B of a given collection of numbers N can be defined by a certain collection of Boolean numbers B. It is natural to choose only those collections of Boolean numbers that are already stored in the machine's memory. We will describe how this choice is made in Section 4. Here we must only say that, in addition, in the memory is always stored a collection of Boolean numbers containing only ones (see also §3.5.) and defining the entire subset. Breaking up the subset $N_B \subset N$ into lots means that we will only apply the algorithm of breaking up into lots on this subset and not on the entire collection of numbers; and re-code into Boolean numbers (in accordance with the thresholds found) only the numbers of this subset.

Thus in its general form the threshold operator has two inputs, into which one collection of numbers N and one collection B_0 of Boolean numbers are inputted; as output, it can produce several collections of Boolean numbers $(B_1, B_2, \ldots B_k) = H(N, B_0)$.

Let us write down the collections of Boolean numbers at the output of the threshold operator (for simplicity we will exclude the case when the input is a collection of numbers with the dimension of slope or orientation). Let $t_1, t_2, \ldots, t_{k-1}$ be threshold values, calculated with the help of the algorithm of breaking up into lots.

Suppose $t_0 = 0$, $t_k = 63$. Then we can write:

$$b_j^i = \begin{cases} 1, & \text{if } t_{j-1} \le n^i \le t_j \text{ or } b_0^i = 1 \\ 0, & \text{otherwise} \end{cases}$$

$j = 1, 2, \ldots, k$.

3.5. Operators of breaking up the pictures into fragments.

3.5.1. Function. A special type of operators are operators that break the pictures up into fragments. These operators put each input picture into correspondence with several output picture-fragments. Use of operators of breaking up the pictures into fragments allows us to process individual parts of any picture, or parts of parts. As a result the program is able to describe classes of pictures, based both on their general characteristics and their small, local elements.

The examples of work of two operators of this type, *breaking up by connectedness* (*S*) and *breaking up by borders* (*J*), are given on Fig. 1 and Fig. 12. In the Appendix on p. 72 are given some problems for which such operations are necessary. In particular, the operator *breaking up by connectedness* is used to describe classification principles of the majority of problems. The other operator, *breaking up by borders*, is used to break up pictures in problems #13 and #39. Problem #15 is an example of a case where both operators make part of the description of the classification principle.

On p. 78, examples of problems (#43, #44, #45) are given for which it is appropriate to introduce one more operator of this type: *breaking up of lines by branching nodes*.

3.5.2. Main definitions. Measuring and drawing operators put into correspondence one object of the output collection with each individual object from the input collection. The threshold operator is not applicable to individual objects; nevertheless, the correspondence between objects of the input collection and the objects of each of the output collections of this operator is defined: $\left(b_1^i, b_2^i, \ldots, b_k^i\right) = H\left(n^i, b_0^i\right)$. There is no such one-to-one correspondence between the input and the output in case of the operators *breaking up into fragments*. Indeed, such an operator divides each picture p_1^i of the input collection P_1 into several picture-fragments $p_2^{ij} \in P_2$; as a result, the output collection contains more pictures than did the input collection[3]. Since we can apply the same procedure to the picture-fragments as we did to the initial pictures (each picture-fragment

[3] If it turns out that a certain operator *breaking up into fragments* "divides" each picture of the input collection into only one fragment, then we say that the given operator is not applicable to the given collection of pictures - see §3.5.3.

$p_2^{ij} \in P_2$ in its turn may be divided with the help of one more operator *breaking into fragments* into several fragments $p_3^{ijk} \in P_3$), it can be said that use of the operator *breaking up into fragments* produces objects of a different level.

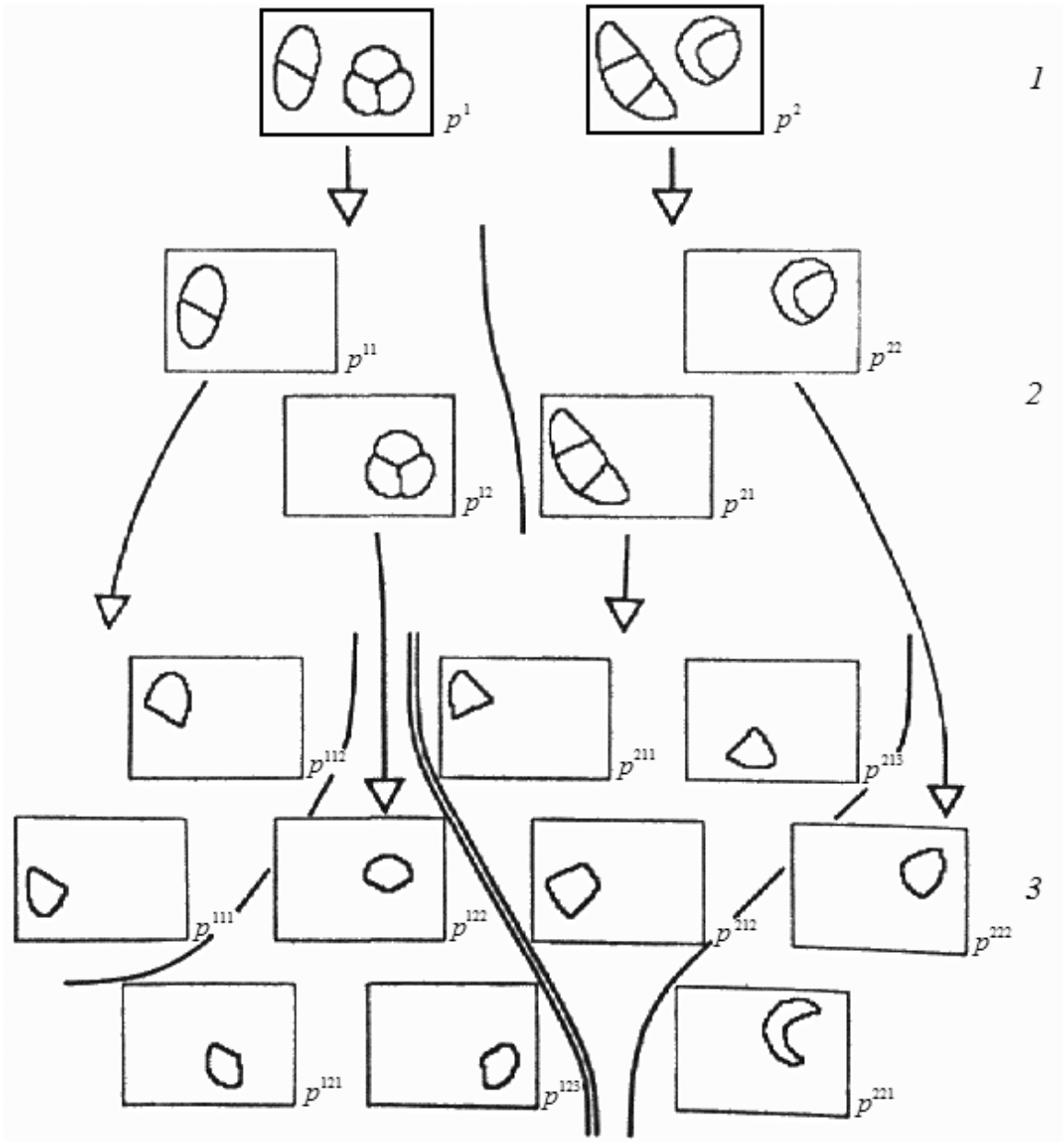


Figure 11. Three levels of pictures that result from successive application of the operators *breaking up by connectedness* and *breaking up by borders* to two pictures p^1 and p^2 of the first level.

Let us define the *level of objects*. We will say that two collections belong to the same level if among the objects of the two collections there exists one-to-one correspondence. The levels of objects can differ by number. The collection of training pictures belongs to the objects of the first level. Application of one of the operators *breaking up into*

fragments to the collection of pictures of the first level produces a collection of pictures of the second level and so on (Fig. 11).

Let us look at the interrelation of objects of different levels. Let p_1^i be pictures of a certain collection P_1 of the first level. Application of one of the operators *breaking up into fragments* to these pictures gives the collection of picture-fragments $p_2^{ij} \in P_2$ of the second level. Here the pictures p_2^{ij} with variable values of the top index j and a fixed value of i are parts of the same picture p_1^i. In other words, an operator *breaking up into fragments* defines the separation of the sum total of objects of the output collection of pictures P_2. Application of the next operator *breaking up into fragments* to picture-fragments p_2^{ij} will yield a collection of pictures $p_3^{ijk} \in P_3$ of the third level. On these pictures there will already be two levels of division: pictures of the third level p_3^{ijk} with variable indices k but with constant indices i and j are parts of the same picture p_2^{ij} of the second level; while pictures with variable values of j and k and a constant value of i are parts of the same picture p_1^i of the first level. Thus, the order of division into fragments defines the structure of breaking-up of a certain level.

Let us recall that operators that transform objects inside one level (drawing, measuring, *threshold*) do not change the objects' indices. They do not use the information about previous divisions for their work, which is coded in the indices. In subsequent sections we will describe operators that we will need, besides the objects themselves in input collections, the information about which fragments were received from which pictures (or parts of pictures). This information must be contained in the collection; therefore we must define the concept of "collection of objects" more precisely.

A collection of training pictures is a collection of objects of the first level. A collection of objects of the m-th level is the sum total of objects that appears at the output of a certain operator as the result of the application of this operator to a collection of objects and for which the entire structure of $m - 1$ levels of divisions is defined.

Further on, while speaking about interrelations of objects of different levels we shall not write down all the top indices for objects of different collections; instead we will indicate only the number of the level and those indices that are relevant for the given relations.

3.5.3. Structure. A collection of pictures of a certain level is given as input to the operator *breaking up into fragments*. At the output, this operator produces the collection of pictures of the next level. In addition, the operator breaking up into fragments enters into memory a collection of Boolean numbers (of the same levels as the pictures at the output of the operator), consisting only of ones. Consequently, each operator *breaking up into fragments* has one input and two outputs (see the list on p. 8).

The operator *breaking up into fragments* may not work on a certain collection of pictures, failing to produce objects of the next level. This happens in the following cases: a) the pictures do not break into fragments with the help of this operator, that is, the attempt produces only one fragment from each picture of the input collection; b) the total number of parts happens to be > 125; c) at least one of the pictures of the training collection contains > 62 parts.

The last two stipulations are made because 1) the size of collections is restricted in the program, and 2) only values from 0 to 62 are permitted for natural numbers and, in particular, for the number of fragments on each picture.

The last restriction does not mean that a certain operator *breaking up into fragments* cannot divide pictures into more than 62 parts. It is necessary that the entire chain of successive divisions of a picture from the training collection give not more than 62 parts. It can happen, though, that each of the operators breaking up into fragments divides each of its input pictures into a smaller number of parts.

Algorithm of breaking up by connectedness S. In the operator *breaking up by connectedness* the internal procedure S^* is used — the isolation of one connected area S^*p from the initial picture p. Multiple application of this procedure divides a picture into separate connected areas. The procedure S^* consists of individual scans of a picture, during which all its columns are successively analyzed. Some elements of the picture are copied onto a separate area p1. Such scans can be repeated several times, moving first from left to right and then from right to left.

Initially $p_1 := \phi$, $p_2 := p$.

1. Moving from left to right, we go over all the columns of picture p_2 until we find the first non-empty column.
2. In this non-empty column (first on the left), we find the highest black point. We add it to picture p_1 and erase it from picture p_2.
3. In this column of picture p_2 we find all the elements connected (in the sense 2) to picture p_1. We add all these elements to picture p_1 and erase them from picture p_2.
4. In case we still have not arrived to the edge of the picture and the next column of $p_1 \cup p_2$ is not empty, we pass to the next column of picture p_2 and return to step 3 of the procedure.
5. In the contrary case we reverse the direction of scanning the columns and, starting from the last column of the previous scanning session, return to step 3 of the procedure.
6. The algorithm finishes its work if during a certain scanning in one direction not a single element was added to picture p_1. As a result the connected area S^*p (isolated from picture p) lies in p_1, and the remainder $p_2 = p \setminus S^*p$.

Algorithm of breaking up by borders J. Analogously to breaking up by connectedness, the algorithm of breaking up by borders uses an internal procedure J^* — isolation of one part $p_1 = J^*p$ from the initial picture p. By a repeated application of this procedure to the remainder p_2 the next part of the picture is isolated. The procedure is repeated until all parts are isolated and p_2 becomes empty ($p_2 = \phi$).

Procedure J^* is built with the help of the already described operations C, F, S^*, as well as operation W_1 — spreading of the picture.

The spreading of the picture consists in the following: we add to the initial picture p elements that have at least one black point neighboring with p (in the sense 1).

The expressions for the isolated part of the picture $p_1 = J^* p$ and the remainder p_1 are the following:

$$p_1 = p \cap W_1 (S^* (Cp \Delta W_1 (Fp \setminus Cp)));$$

$$p_2 = p \cap W_1 (\mathrm{p} \setminus \mathrm{S}^* (Cp \Delta W_1 (Fp \setminus Cp))).$$

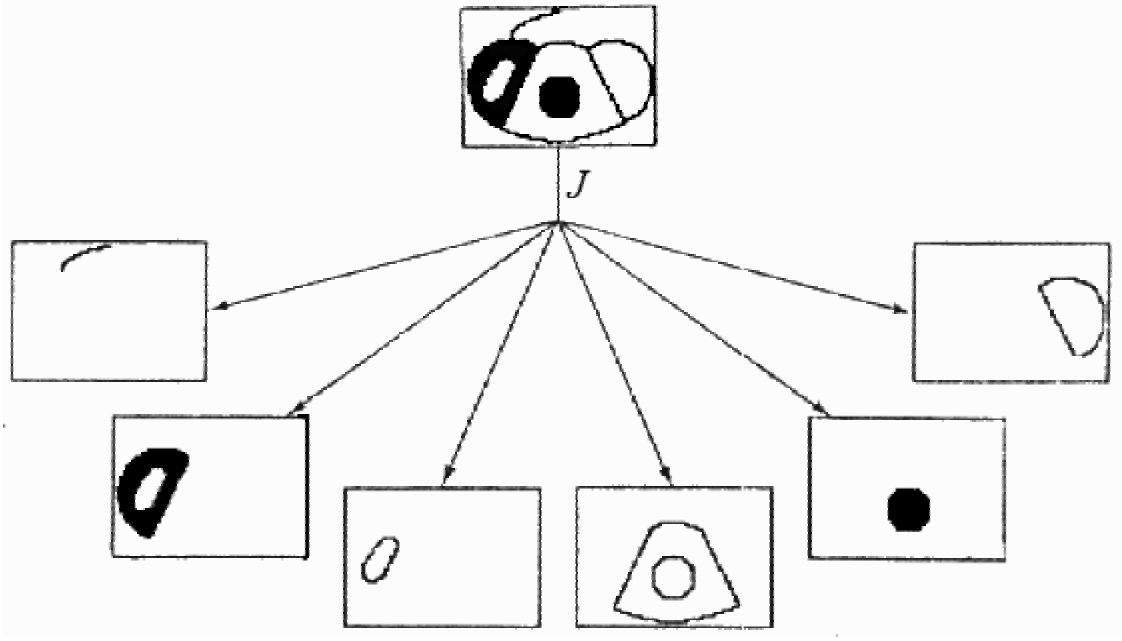


Figure 12. An example of work of the operator *breaking up by borders J*.

Algorithm of breaking up of lines by branching nodes K. As internal procedures, the operator *K* uses the operation of breaking up lines by connectedness (*S*), the procedure of thinning lines (*V*), and the operation W_2 — spreading of picture. The last operation differs from W_1 which was used in the algorithm of breaking up by borders in that in W_2 such elements are added to the initial picture that have at least one black point neighboring with *p* (in the sense 2).

The steps of the algorithm.

1. Apply the procedure of thinning of lines to the initial picture *p*: $p_1 = Vp$.
2. Erase all the node points (points having more than two neighbors — in the sense 2) from picture p_1 and draw them in p_2.
3. In picture p_1 choose among all node-points (that is, points of figure $W_2 p_2 \cap p_1$) the isolated points (those that do not have neighbors in the sense 2). Erase these points on p_1 and add them to p_2.
4. For each connected area in p_2 calculate the number of points neighboring with it in the in p_1; if it is fewer than 3, erase this area from p_2 and add it to picture p_1.
5. Break up picture p_1 by connectedness and add to each of thus obtained pictures all the areas of picture p_2 that are connected with it.

3.6. Operator comparison. 3.6.1. Function. There exist problems in which the distinguishing rule is formulated in a form of certain quite simple functions of continuous parameters. As concrete examples, let us look at problems #35 and #36. In #35 the classes differ by the "blackness" of figures in the pictures -- not by the total number of the black points, but rather by their average density. In order to define this density it is sufficient to calculate the number of black points in the picture (applying the measuring operator *area* directly to the picture), measure the area of the figure (by doing convex hull filling and then applying the operator *area*) and divide the first result by the second one. In this way the elementary operators such as *area* and *convex hull filling* allow us to single out the parameters relevant for constructing the distinguishing rule. Now we only have to calculate their ratio. In problem #36 we find an analogous situation. Here the program on a certain stage will build in each picture a region occupied by small points and measure its area. It will separately measure the area of the region bounded by lines (in §3.8 we will say which "complex operators" are needed for this; the solution of this problem will be analyzed in detail in §5.5). It can easily be seen that neither one nor the other parameter can be broken into lots; however, both of them are necessary for the description of the distinguishing rule. Indeed, in this problem the classes differ from one another in that in the pictures of one class the lines enclose bigger area that do the points, whereas for the pictures of the other class the contrary is true. It can be noticed that here the ratio of these two areas can be broken up into lots, and the pictures of each class will end up in their own lot. In another problem (#34) the comparison of the area of the figure (after the contour filling) with the area of its convex hull will allow us to distinguish figures with bigger and smaller concavity. Problems #37 and #41 require comparison of other parameters: slope, coordinates. It should be noticed that not any pair of parameters can become arguments of distinguishing rules. People never compare the slope with the length of lines, but the comparison of two elongated fragments of a picture allows us to define the angle between those fragments.

Thus, the examples show that an operator is needed which would allow us to compare different collections of numbers of the same dimension and level. In principle, it could be possible to construct such an operator which on the basis of two input collections of numbers of the same dimension would construct a single output of numbers. This output would be stored in memory according to the general scheme; then the attempts of applying the *threshold operator* to this collection of numbers (or to certain subsets of this collection) using *breaking up into lots* would lead (or not) to the coding of this collection of numbers in Boolean numbers. Initially [5,6], the idea was to construct the operator *difference*[1] precisely in this form. Of course, with such an approach we would need a lot of space to store all the collections of numbers coming out at the output of operator *difference*. However, this is not what is important here. As it has already been said, the main principle of the program consists in that all the collections of objects, kept in machine's memory, are given as input to all the operators applied to the objects of the given type. This scheme of work makes it natural to input to operator *difference* its own

[1] Indeed, it was proposed to introduce several such operators. Different variants of operators of this type were called *difference I, ratio* [5], *difference II, difference III (or difference module)* [6]. To a certain extent, the names reflected the character of transformations performed by these operators. As it will be seen further, the single operator *comparison*, described in this section, is a generalization of the first three operators.

output collections kept in memory, in order to calculate difference of difference, etc. The possibility of such superposition of the operator *difference* will allow the program to easily construct quite complex functions of multiple variables. For instance, as shown by experiments, people as a rule are not able to formulate distinguishing rules in such terms. Consequently, distinguishing rules of this type should always be qualified as superstition. In order to avoid this, we should forbid the input of the collections of objects appearing at the output of the operator *difference* into the same operator. In other words, whereas it is useful to keep the initial collections of numbers in machine's memory, since each one of them may be analyzed by various operators, collections–outputs of the operator *difference* should only be broken up into lots. This is why we eventually adapted the following scheme of work: 1) two input collections of numbers N_1 and N_2 of the same dimension are transformed into a certain intermediate collection of numbers which is not stored in memory, 2) a procedure analogous to the *threshold operator* should be immediately applied to this last collection — the collection (or its subset defined by a certain collection of Boolean numbers B_0) is broken up into lots and coded by a collection of Boolean numbers $B_1, B_2, ..., B_k$. These collections of Boolean numbers are outputs of the operator and are stored in memory. Thus, the operator *comparison* must have three inputs N_1, N_2, B_0 and up to four outputs $B_1, B_2, ..., B_k$:

$$(B_1, B_2, ..., B_k) = \mathrm{R}\ (N_1, N_2, B_0).$$

The character of comparison of the input collections of numbers should be defined by their dimension. Thus, for the natural numbers (outputs of the operator *number of fragments,* see §3.7) it is natural to calculate their difference; for the slopes, the difference of slopes should be calculated; for areas, lengths and sizes of axes, it is natural to calculate the ratio of the corresponding parameters (or, on the logarithmic scale, their difference). We should pay special attention to the case when the input collections of numbers are coordinates of centers of gravity. By applying the *threshold operator* we were trying to break up into lots independently the *x*-coordinates n_{x}, and the *y*-coordinates n_{y}; the comparison of solely *x*-coordinates or solely *y*-coordinates does not seem appropriate. Indeed, the comparison of coordinates (for example, the centers of gravity of two different figures in a picture) should make it possible to formulate a statement about the mutual orientation of these figures. Apparently such parameters as the distance between the centers of gravity and the direction from one of the centers of gravity to the other corresponds more to human perception than do the differences of solely *x*-coordinates or solely *y*-coordinates. Consequently, we may conclude that in case of coordinates, the procedure of breaking up into lots by the operator *comparison* should have as coordinates not two collections of numbers, but rather two 2-dimensional vectors $\left(n^i_{x1}, n^i_{y1}\right)$ and $\left(n^i_{x2}, n^i_{y2}\right)$ constructed out of the corresponding collections of numbers. Those vectors are first expressed as two intermediate collections of numbers — distance and direction — each one of which is then broken up into lots.

The variant of the operator *comparison* introduced here is in fact a special case of a more universal operator. To illustrate the situation, let us look at the Problem #38. Here the

classes differ in that in the pictures of the class to the right the segments composing the elongated figure have a transversal orientation, whereas in the pictures to the left the corresponding orientation is longitudinal. It is easy to see that it is impossible to describe this distinguishing rule in terms of already existing operators (that is, to construct a complex operator from some elementary ones). Indeed, we can measure the slope of the figures (more precisely, the slope of the longitudinal axis of the convex hull of the pictures), and the separation of the pictures into fragments would allow us to measure the parameters (including the slope) of the individual segments. The difference of these slopes could give us the distinction — in the pictures of the left class it is 0° for all fragments, whereas in the pictures of the right class it equals 90°. However, the operator *comparison* described above is not applicable here. The collection of numbers characterizing the slope of the figures belongs to the first level, and the collection of numbers characterizing the slope of the segments, to the second level. It is obvious that in the general case we may have to compare the collections of numbers not necessarily of the contiguous levels, but of any pair with the numbers *l* and *m* (where *l* is smaller than or equal to *m*) in the sequence of levels.

Thus it is necessary to expand the area of the applicability of the operator *comparison* so that it would allow us to compare collections of numbers of the same dimensionality but, in the general case, of different levels. For this it is enough to stipulate that for all fragments p_m^{ij} of picture p_l^i each of the numbers n_m^{ij} (characterizing these fragments) would be compared with the same number n_l^i (characterizing picture p_l^i). All the above considerations about how (depending on the dimension of the input collections of numbers) these comparisons are to be made, and in which form the results of the work of this operator are to be stored in memory, remain relevant also for the case when the collections of numbers belong to different levels. There exists only one exception. The collections of numbers belonging to different levels should not be compared in case they represent the outputs of the operator *number of fragments* (natural numbers). It turns out that no problem (easy for a person) can be constructed in which such an operation must be used in the description of the distinguishing rule. Moreover, there accumulates in the memory a large quantity of such collections of numbers and all sorts of combinations of such collections — a rich source of “superstitions”.

In the final form the operator comparison has three inputs — the collection of numbers N_{l1} of the *l*-th level, the collection of numbers N_{m2} of the *m*-th level (*l* is smaller than or equal to *m*) and the collection of Boolean numbers B_{m0} of the *m*-th level and can form at the output (similarly to the *threshold operator*) several collections of Boolean numbers of the *m*-th level.

$(B_{m1}, B_{m2}, \ldots, B_{mk}) = R\,(N_{l1}, N_{m2}, B_{m0})$.

3.6.2. Structure. *The First Step.* From the input collections of numbers N_{l1} and N_{m2} ($n_{l1}^i \in N_{l1}$, $n_{m2}^{ij} \in N_{m2}$, $l \le m$) we form collections of numbers N_1 and N_2 ($n_1^j \in N_1$, $n_2^j \in N_2$) of the *m*-th level (which are not stored in memory but become arguments of the next step of the algorithm).

In case $l = m$ we suppose that $n_1^j = n_{l1}^j$, $n_2^j = n_{m2}^j$..

In case $l < m$ the collection N_1 is formed in the following way: $n_1^j = n_1^{ij} = n_{l1}^i$, for all j corresponding to the given value of i.

In case when the inputs are coordinates, both components of vectors are formed in the analogous way $\left(n_{x1}^j, n_{y1}^j\right)$ and $\left(n_{x2}^j, n_{y2}^j\right)$.

The Second Step. Two collections of numbers (or two vectors of coordinates) are transformed into an intermediate collection of numbers N_3. The character of transformation of the input collections of numbers is defined by their dimensionality.

a. Dimension — slope

N_3 is calculated by the formula

$$n_3^j = \begin{cases} \left(n_2^j - n_1^j\right) \bmod 8, & \text{if} \quad n_1^j \neq f \text{ and } n_2^j \neq f \\ f, & \text{otherwise.} \end{cases}$$

The same dimensionality is ascribed to the N_3 as to the input collections of numbers.

b. Dimension — natural numbers, area, length, size of axes.

First $d^j = 62 + n_2^j - n_1^j$. In principle, the range of changes of d (from 0 to 124) can be twice as big as permitted (0 - 62), even though it will rarely be the case in real problems. In order to "fit" these values of difference into the necessary range, the numbers of the intermediate collection are calculated by the formula

$$n_3^j = \begin{cases} 0, & \text{if} \quad d^j - \delta \leq 0, \\ d^j - \delta & \text{if} \quad 0 < d^j - \delta \leq 62, \\ 62, & \text{if} \quad d^j - \delta > 62, \end{cases}$$

where $\delta = (\max\{d^j\} + \min\{d^j\} - 62) / 2$.

The same dimensionality is ascribed to N_3 as to the input collections of numbers.

c. Dimension — coordinates.

For all values of j for which $n_{x1}^j, n_{y1}^j, n_{x2}^j, n_{y2}^j \neq f$, the values

$$\left(l^j\right)^2 = \left(n_{x2}^j - n_{x1}^j\right)^2 + \left(n_{y2}^j - n_{y1}^j\right)^2$$

are coded on the logarithmic scale analogously to the coding of the outputs of the measuring operator *length of lines*. For all the other values of j we suppose $n_l^j = f$. We ascribe thus to the obtained intermediate collection of numbers N_l the dimension *length*.

In order to calculate direction we use the formula

$$\tan\psi = \left(n_{y2}^j - n_{y1}^j\right) \Big/ \left(n_{x2}^j - n_{x1}^j\right).$$

We distinguish among 16 gradations of direction (see Fig. 6). In case when at least one of the four numbers $n^j_{x1}, n^j_{y1}, n^j_{x2}, n^j_{y2}$ is equal to *f*, or the distance between the coordinates is small ($n^j_l \le 2$) we suppose $n^j_\psi = f$.

We ascribe the intermediate collection of numbers N_ψ the dimension *direction* .

The Third Step. The intermediate collections of numbers (or some of their subsets defined by the input collection of Boolean numbers B_0) are broken up into lots and coded by collections of Boolean numbers, analogously to the way it is done in case of the *threshold operator*.

3.7. Operators of the logical block. 3.7.1. Preliminary observations. *Breaking up of the pictures into lots* is the only type of operators that form from the object of one level to collections of objects of next levels. Three of the four operators we shall now describe are doing the opposite: applied to the collections of objects of deeper levels (with a bigger number), they supply objects for the previous levels (Fig. 13).

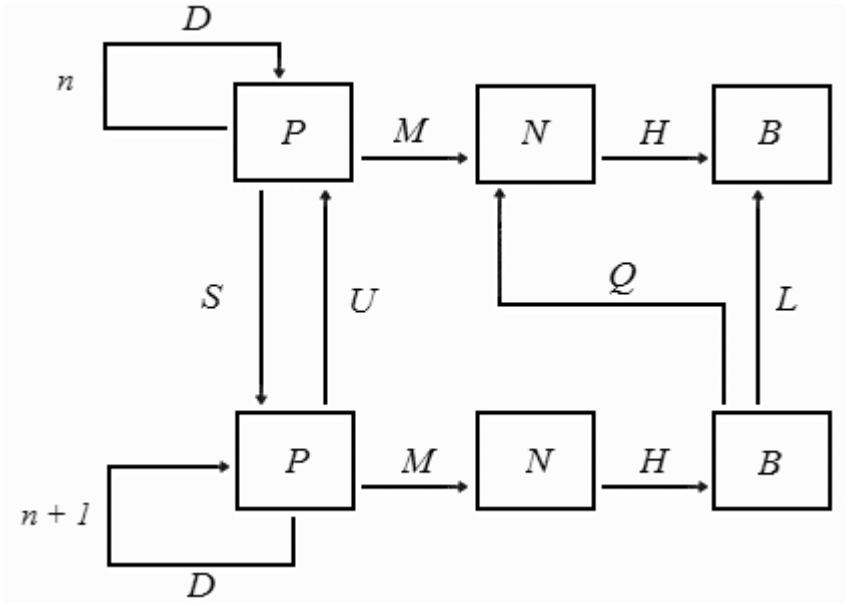


Figure 13. *The scheme of transformation of objects of two consecutive levels.* Rectangles — types of objects (*P*: pictures; *N*: numbers; *B*: Boolean numbers). Types of operators transforming one object into others are shown by arrows (*D*: *drawing operators, M*: *measuring operators, S*: *separation of pictures into fragments, U*: *union, H*: *threshold operator, Q*: *number of fragments, L*: *logical operator*).

It must be noted that a single application of any of these operators to any collection of objects of the *m*-th level supplies the output collections not only to the *m*-1-th level but also to all the previous levels, from the 1-st level to the *m*-1-th level. We will not specifically mention this fact while describing the work of the operators; we will only describe the character of the output collections of objects of one of the previous levels (*l* < *m*); nevertheless, it should be remembered that there can be several such levels.

3.7.2 Decision-making operator. The *decision-making operator* can be applied only to collections of Boolean numbers of the first level. The work of this operator consists only

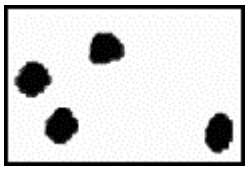
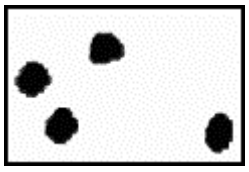
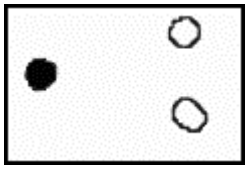
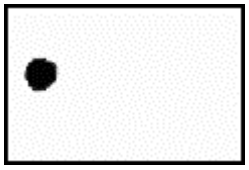
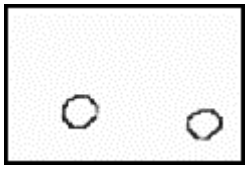
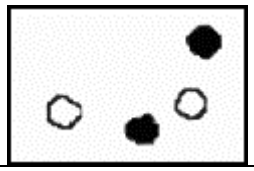
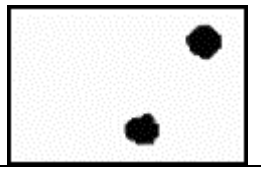
in checking whether the given collection of Boolean numbers is distinguishing. If this is the case, then the number of this distinguishing collection of Boolean numbers of the first level is printed (see §4), the problem is considered solved and the machine stops working. This stop can be considered the output of this operator.

It has to be noted that the information about breaking up into classes the pictures of the training collection is not used anywhere else in the program. All other operators are working with the sum total of objects of a collection as with a whole.

3.7.3. ***Operator number of fragments. Function and structure***. The pictures can differ by the number of fragments (Problems #12 and #13) or by the number of fragments of a special type (Problem #14). More complicated principles of classification, into which the number of parts enters as a significant parameter, are also possible (Problem #15).

A subset of pictures characterized by a certain property is defined by the corresponding collection of Boolean numbers. That is why in order to calculate the number of fragments of a certain type this collection of Boolean numbers should be given as input to this operator. Consequently, the operator *number of fragments Q* has at the input a collection of Boolean numbers B_m characterizing those fragments and at the output, the collection of numbers of the previous level $N_l = QB_m$. Table 1 illustrates the work of this operator.

Table 1

Input		Output of operator *num. of fragments*	Outputs of *logical* operator				Output of operator *union*
			a	b	c	d	
	1 1 1 1	04	1	0	0	1	
	0 1 0	01	0	1	0	1	
	0 0	00	0	1	1	0	
	1 0 1 0	02	0	1	0	1	

Let $b_m^{ij} \in B_m$, $n_l^i \in N_l$, $l < m$, then the number n_l^i of the output collection of objects of the operator *number of fragments* is calculated by the formula $n_l^i = \sum_j b_m^{ij}$, where the summation is done for all *j* corresponding to the given value of *i*.

We ascribe to the output collection of numbers the dimension *natural numbers*.

3.7.4. Logical operator. Function and structure. To give the description of distinguishing rules based of the characteristics of fragments in the pictures, it is advisable to make use not only of the numerical functions of the collection of Boolean numbers describing the fragments (that is, counting the number of fragments), but also of some logical functions. Such functions may be useful for describing distinguishing rules in many problems. As typical examples, we can cite the classification principles in Problem #27 ("all the fragments of the pictures of the left class are situated in the middle of the raster") and in the Problem #30 ("in the pictures of the left class there exists at least one small square").

Thus, *logical operator* is applied to collections of Boolean numbers characterizing fragments (number of level > 1) and supplies objects to the previous level. To each input collection, it puts into correspondence four output collections of Boolean numbers of the previous level, corresponding to the following four logical functions: a) to all fragments of the picture in the collection correspond ones, b) not to all fragments of the picture in the collection correspond ones, c) to all fragments of the picture in the collection correspond zeros, d) not to all fragments of the picture in the collection correspond zeros. An example of functioning of the *logical operator* is given in Table 1.

Analogously to the operator *number of fragments*, the Boolean numbers $b_{la}^i, b_{lb}^i, b_{lc}^i, b_{ld}^i$, of the output collections of objects of logical operator are calculated by the formulae

$$b_{la}^i = \bigwedge_j b_m^{ij}, \quad b_{lb}^i = \bigvee_j \overline{b}_m^{ij}, \quad b_{lc}^i = \bigwedge_j \overline{b}_m^{ij}, \quad b_{ld}^i = \bigvee_j b_m^{ij},$$

where the conjunction and disjunction are taken for all *j* corresponding to the given value of *i*.

3.7.5. The operator union. Function and structure. *Union* performs an operation which is inverse to division into fragments — from a certain number of fragments of a picture the operator *union* forms a new picture. The output collection of objects of this operator belongs to the previous (in relation to the input one) level. The information about which fragments are to be unified comes in the form of a collection B_m of Boolean numbers characterizing the fragments to the second input of the operator *union*: $P_l = \mathrm{U}\,(P_m, B_m)$.

An example of functioning of the operator *union* is given in the Table 1.

Pictures $p_l^i \in P_l$ at the output of the operator *union* are constructed from pictures-fragments $p_m^{ij} \in P_m$ by the formula

$$p_l^j = \bigcup_j p_m^{ij},$$

where the operation *union* is done for all *j* corresponding to the given value of *i* such that $b_m^{ij} = 1$.

3.8. Complex operators. People are able to solve various problems of classification of geometric objects. The analysis shows that people often use the same terms to describe completely different objects. It may seem that such terms correspond to the most general (from the human point of view) characteristics of these objects. Consequently, for modeling human behavior it is sufficient to create special blocks, isolating these characteristics; after that it will be easy to build a system capable of classifying objects in "human" terms. It turns out, however, that the existing mathematical definitions of the corresponding concepts (such as angle, straight line, etc.) are not always applicable. Indeed, people often give the name of a 'triangle' to a figure whose sides are not straight and whose angles are rounded. Thus it is natural to try to find more general definitions.

Similar attempts are characteristic for the linguistic approach to the problem of pattern recognition. Each picture is described in terms of the local characteristics ('angles', 'curves', 'nodes', etc.), common for all figures and all problems [7]. Probably this attempt can prove successful in a limited domain, for example, in the problem of recognition of letters and written signs. Unfortunately, if we pass to a wider class of problems we will immediately be confronted with the following characteristic of human language: the same terms in different problems have different meaning (that is, one term characterizes different classes of objects). In other words, it is not the broadening of a term's meaning that leads to its universality, but the possibility to put this term into a relationship with a concrete situation.

In this sense, the example of the Problem #46 is significant; here the "human" principle of classification should sound something like this: "Pictures contain straight and curved line segments. To the left class belong the pictures where the curved line segments make a straight line." It follows from this example that no matter which definition of the straight line we try to give, it will not be applicable to both the first and the second phrases of this classification. (Of course, we could leave the term 'straight line' only for the mathematical abstraction, and replace its "colloquial" usage by "stricter" expressions such as "point lying close to a straight line". However this will not improve the situation, since now *these* expressions will take different meaning on different levels of the description of the distinguishing rule.)

It seems that such vagueness of human language creates only confusion and inconvenience at the attempts of any kinds of formalization, in particular, in building the formal language of picture description. However, it is precisely this semantic vagueness of the terms of human language that leads to their universality. As a result, the same semantic constructions, whose structure is uniquely defined by the terms composing them, may be used to describe very different situations.

If we attempt to introduce this characteristic of human language into the language of the program, we will arrive to the following requirement. The program must be able to form terms (operators) whose meaning would depend on the situation (that is, on the objects which are given as input to the operators). We will try to demonstrate that the proposed language of operators possesses, to a certain extent, the required flexibility. From elementary operators, the program can build complex operators whose behavior will be

defined by the concrete collection of input objects. The flexibility of this language is due mainly to the procedure of breaking up into lots. Let us see an example. There is no operator in the program that corresponds to the concept “big”. However, such an operator is constructed from the sequence of two operations: measuring of an area and then transformation of this area (with the help of the *threshold operator*) into Boolean numbers. Naturally, there are no *a priori* big or small figures; it all depends on a concrete situation. It is precisely the operation of breaking up into lots that takes into account the situation, while choosing the threshold separating the big areas from the small ones. Moreover, a concrete situation (when the area cannot be broken up into lots) may make the application of such terms meaningless.

The program can indeed give different meanings even to such terms as “contour” and “convex”, which are defined, as it seems, by the strictly defined structure of the corresponding operators (we are talking about that part of the drawing operators which gives as output Boolean numbers corresponding to the statements “the input picture is a contour one”, etc.) Let us analyze this in more detail. The Boolean outputs of the drawing operators *contour isolation* and *convex hull filling* are strictly defined by the structure of these operators and do not depend on the situation, that is, on which collection of pictures is given as input to these operators. It seems that the terms “contour” and “convex” have a narrow sense which can change only if the structure of these drawing operators changes. However, in the program there exists a way around it. The program can replace these strict Boolean operators by the corresponding (and even more universal) complex operators. Of course, the drawing operators *contour isolation* and *convex hull filling* will be used here as components. However, in forming of a Boolean output, we will use not the strict scheme of comparison (whether the picture at the output of a drawing operator coincides with the input picture, or not), but the procedure of breaking up into lots. For example, it is possible to build a complex operator that would compare the areas of a figure and its convex hull (breaking up into lots the ratios of these areas with the help of the operator *comparison*). It is natural to say that the pictures for which this ratio is close to 1 are convex, and all the other pictures are concave. The boundary of such a division will, of course, depend on the situation: certain pictures will be perceived now convex and now concave in different situations. And if all the possible values of this ratio are found with an almost equal frequency, it means that this ratio cannot be broken up into lots: in the world where all the intermediate forms are possible there is no sense in dividing the figures into convex and concave. There exist situations in which the above-described procedure of separating a picture into convex and concave figures will give the same result as will a strict Boolean operator. This result will mean that this operator turned out to be adequate to the collection of pictures it received as input. Analogously, we can build a complex operator for the term “contour”.

Thus, in a strict Boolean operator the meaning of a term depends on the construction of the drawing operator and does not depend on the situation. In the corresponding complex operators there appears universality — a strong dependency on the situation. As a result, the work of these operators is not defined anymore (within certain limits) by the structure of elementary drawing operators making part of these complex operators. Consequently, the language of the program has the following characteristic: the terms in which objects are described are defined better by the sum total of these objects than by the elements of the language itself.

Now let us return to Problem #46 which we have analyzed earlier. Unfortunately, the present variant of the program loses much of its value because it lacks the point-by-point analysis of lines (in particular, it does not take into account the slope and the direction of the line in a point). However, the program is able to create a "definition" of a straight-line segment — distinguishing straight lines from curved ones — in terms of drawing operators it already possesses. A straight-line segment is a figure which at the same time is contour and convex.[4] As we have seen, the program can build varied (defined by concrete circumstances) formulations for the concept *contour* as well as for the concept *convex*. Thus it is natural that on different levels of description of the distinguishing rule of Problem #46 the program is able to build different operators in accordance with those variations which people introduce into the meaning of the term "a straight line segment" (see the solution of Problem #46 on page 73).

Another type of complex operators where the application of the procedure *breaking up into lots* makes the behavior of these operators to be dependent on a concrete situation is complex drawing operators. It may seem that a program containing only three elementary drawing operators (see Fig. 1) cannot provide a large variety of picture transformations (even taking into account all the possible superposition of these operators). Let us show in what way the program is able to build a special type of complex drawing operators via previous division of pictures into fragments. As a result of such division, isolation of a subset of fragments, all of which possess a certain characteristic, and subsequent putting of these fragments into a union, we obtain pictures on which some elements (those which do not possess this characteristic) are "erased". The variety of such re-drawings is defined by the problem itself — it depends on the variety of the collections of Boolean numbers characterizing those fragments. Since these collections of Boolean numbers are supplied mostly by the *threshold operator* and the operator *comparison*, a modification of the problem (a modification of the sum total of pictures given as input to the complex drawing operator) may change considerably the character of this operator's work.

Let us look at some examples.

In Problem #36 the fragments of pictures are of two types — points and lines (the length of lines is broken up into lots). Applying the operator *union* to the subset of points and to the subset of lines, we get at the output the pictures of the first level which can be interpreted as outputs of two complex drawing operators (Fig. 14).

In Problem #8 breaking up of the pictures by connectedness, putting into union the contour (white) figures, and subsequent convex hull filling will produce at the output pictures in each of which one elongated (horizontally or vertically, depending on whether the input picture belonged to the left or the right class) figure will be represented. This complex drawing operator makes part of the description of the distinguishing rule in the given problem (see Fig. 19).

In order to construct the distinguishing rule in Problem #45, we need a drawing operator that would eliminate the straight-line segments from the input pictures. Such an operator may be built by successive division of pictures into fragments by branching nodes, finding out with the help of the operator *convex hull filling* which of the fragments are

[4] Another variant: a straight-line segment is a convex figure that has small thickness.

made of straight lines (convex) and which are made of curves (concave), and applying the operator *union* to the concave fragments. (Fig. 14).

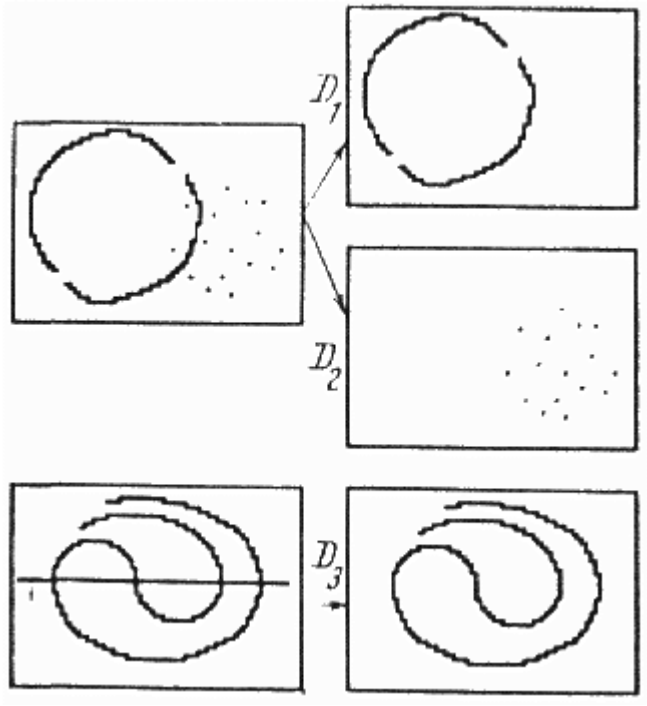


Figure 14. Examples of complex drawing operators

4. Organization of searching

4.1 A general scheme of searching. In solving each concrete problem the goal of the program is to build from some elementary operators a complex (distinguishing/dividing) operator which would put into correspondence to each picture of one class (from the collection of training pictures) the Boolean number 1, and to each picture of the other class, the Boolean number 0. It is clearly disadvantageous to directly generate complex operators one after another, checking whether each one of them is distinguishing/dividing for the given problem. It appears more natural to build them starting from "zero" in form of constantly growing chains of elementary operators, until the necessary distinguishing/dividing operator will be build. On the one side, such a process represents a branching search of ever longer chains of elementary operators. The generating procedure for this process is defined by the list of elementary operators and rules of their combination. On the other side, this process can be regarded as search among collections of objects obtained from the input collection of training pictures as a result of applying to it various complex operators. The goal of such searching is to find the distinguishing collection of Boolean numbers (see §2.1).

If, for simplicity, we limit ourselves to the analysis of the operators that have one input and one output, then we can schematically represent the searching in the form of a graph whose vertices are collections of objects and whose arcs are various elementary operators. The collection of training pictures will be the primary vertex of this graph. It

has to be stressed that, by its structure, a graph is different from a tree; it is characterized by the number of cycles, that is, its vertices can be reached not by a single way, but by many ways. This trait suggests that the object of searching would be not the different paths in this graph (complex operators), but rather its vertices. Indeed, in the process of complete searching of vertices the program will eventually find at least one distinguishing rule, if it exists and can be described in the language of the given set of elementary operators.

Consequently, our task boils down to building a scheme that would guarantee complete searching through all the collections of objects which can be build on the basis of the training collection, via all the possible superpositions of elementary operators. Let us analyze some possible approaches to this problem. One of them consists in reducing the graph of searching to a tree (for trees there exist simple schemes of searching). For this, in the process of searching we can provisionally say that each one of the ways that in reality cross at a common vertex ends at a separate vertex. In this way we would ignore the cycles. Unfortunately, such an approach is unacceptable not only because it can lead to an increase of the work time due to repetitions, but rather because it transforms our final graph into an infinite tree (since it contains closed contours).

An alternative approach may consist in constructing, instead of the generation procedure for a finite list of elementary operators, a fixed (though, perhaps, somewhat cumbersome) universal scheme of search of the graph's vertices and introducing this scheme into the program. However, in each concrete problem there appears to be a large number of equivalent (only in the given problem) complex operators — operators that give the same collections of numbers at the output. This leads to a considerable simplification of the graph. Examples of such simplification of graphs constructed from drawing operators are given in the next section. More explicit and varied examples can be found in §5.5, where we describe the behavior of the program working on problems. Consequently, it is desirable to introduce search without repetitions, corresponding to the reduced (in correspondence with the given problem) graph.

For this there exists a method, which we are partly using in the described program. The method consists in the following: the subsequent collection of objects, corresponding to the subsequent vertex, is compared with all the collections stored in memory. If it is not new (a repeated passage by this vertex, while following another path), then it is not stored in memory; this way, the number of objects in memory grows only when we hit a new vertex. Such a searching scheme allows us to pass each arc of the graph only once.

Such a procedure for storing new collections of objects in memory as a whole is used in the program for collections of Boolean numbers. It has to be noted that in case of using *drawing operators* and operators *dividing into fragments* such a procedure does not simplify the graph sufficiently; we will describe the methods of further reduction of searching in the next section.

Reduction of searching of drawings. 4.2.1. Specific difficulties. It turns out that the above described scheme of searching cannot be used in case of operations on collections of pictures for the following reasons:

1. A very big memory is necessary to store of the collections of pictures, obtained in the process of solving the problem of searching[5],
2. Much time would be required to see whether the new collection of pictures coincides with any of the stored collections,
3. Too much machine time is required to transform the pictures themselves (operations of redrawing and dividing into fragments).

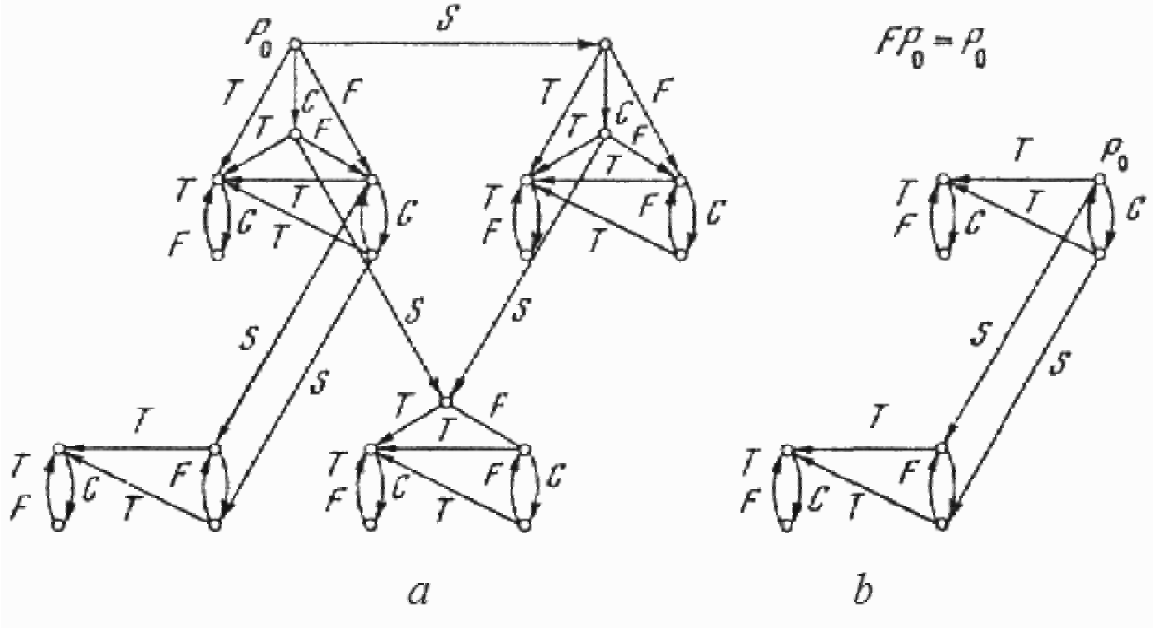


Figure 15. Graphs of possible transformations of collections of pictures with the help of four operators*: contour isolation (G), contour filling (F), convex hull filling (T)* and *separation by connectedness (S)*.

Let us clarify the last point by giving an example. Let us analyze the graph of searching which contains only four operators: *contour isolation, contour filling, convex hull filling,* and *separation by connectedness* (further in this section we will not distinguish between drawing operators and operators of division into fragments; for the sake of simplicity, we will call all of these operations *drawings*). The complete graph generated by these operators is shown on Fig. 15a. The initial vertex of this graph is the collection of training pictures. In total, the graph contains 21 vertices. Since four arcs go out from each vertex (Fig. 15 does not show loops — arcs whose beginning and end are in the same vertex), the total number of arcs equals 84. Thus, working by the general scheme, the program must store in memory collections of pictures corresponding to the vertices of the graph, apply to each of these collections all the operators, and, as a result, go along all the 84 arcs of this graph. However, it is clear that in order to pass by all the vertices of this graph it is enough to travel only along 20 arcs.

[5] The collections of objects, stored in memory, are used in two ways. First (and this is what interests us when we organize searching), the objects are stored in memory so that each new collection of objects could be compared to all the previous collections, to see whether it coincides with any of them; thus the described scheme of searching will be fulfilled. Secondly, collections of objects are necessary for feeding them as input into operators (see §4.3.2.)

As is was said in the previous section, the replacement of the general scheme of searching by a shorter, fixed searching scheme of the vertices of this graph (along certain 20 arcs, chosen *a priori*) is not acceptable. Since the form of the graph depends on the character of pictures of the training collection P_0, it is necessary to give to the scheme of searching some flexibility — an ability to adapt to specific characteristics of each problem. Indeed, the graph of searching will be considerably simplified if, for example, it turns out that in a certain problem the pictures in the training collection do not contain "holes" $FP_0 = P_0$. The corresponding graph is shown on Fig. 15 b. A very simple graph is the outcome if, in the pictures of the training collection, contour ($CP_0 = P_0$) and at the same time convex ($TP_0 = P_0$) figures are given; this is the case, for example, in Problem #25 (see Appendix). Such a graph contains a single vertex, and no re-drawings can change the input collection of pictures.

It has to be noted that with the increase of the power of the machine the significance of these difficulties decreases. Indeed, searching through a large number of superfluous variants can be compensated by higher speed of the machine, and a larger memory capacity allows us to store and then compare all the intermediate results of calculations. This is, approximately, the situation of the block of the program that processes numerical data. All the operators of this block (see list B on page 37) work fast enough, and the processed material does not occupy much space in machine's memory.

4.2.2. Using operational model for reduction of searching. On one hand, the necessity of using a general scheme of searching, and on the other hand, the difficulties described above lead to the following requirement for the organization of the searching of drawings. In order to do the searching of the vertices of a graph of drawings, it is necessary (1) to store, instead of collections of pictures, some other (less cumbersome, allowing to compare pictures for finding the coinciding ones) objects (*images* of collections of pictures), (2) instead of slow drawing operators (which process collections of pictures), to use faster operations with images.

In other words, in order to find a path in a graph of drawings by the general scheme, we propose, instead of collections of pictures P and operators D transforming the pictures, to use a certain operational model; namely, one such that there exist (1) a mapping of the collections of pictures onto the elements (images) Π of the model $P \to \Pi$, (2) a mapping of the operations D onto the operations Δ of the model, such that the condition $DP \to \Delta\Pi$ is always fulfilled.

In the machine implementation the images of the model take up several digits of a single cell; the operations of the model contain a couple of commands. Thus this model, which has the same graph of transformations, allows us to do faster searching of the vertices by the general scheme.

Let us give an example of using this model in the scheme of searching of drawings. Let us look at a single stage of the program's work. A certain collection of pictures P is inputted into the operator D. Before performing the corresponding operation of redrawing, the image Π of this collection of pictures P and the corresponding operator Δ are taken from the memory. The transformation $\Pi_1 = \Delta\Pi$ is done. The element Π_1 is compared to all the images stored in memory. Two cases are possible.

1. Π_1 coincides with one of the images in memory. This means that the application of operation *D* to the collection of pictures *P* will give a collection of pictures that has already been obtained in some other way. That is why the program does not lose time for the senseless application of the operator *D* to the collection of pictures *P*.
2. If the image Π_1 has not yet appeared before, then a) operation *D* is applied to the collection of pictures *P* and b) the image Π_1 is stored in memory.

The reader may easily see that such a scheme, applied to the graph in Fig. 15a, would indeed allow us to do a searching of all its vertices traveling only along 20 arcs and discarding the other 64 arcs after trying them on the model.

From the above scheme of the model it can be seen that the graph of searching is defined by the initial image Π_0 corresponding to the collection of training pictures P_0. All the subsequent images Π of collections of pictures are obtained from Π_0 by standard operations Δ. Since we cannot know beforehand what kind of problem the program will have to solve, we have programmed such a Π_0 that would give the most complete graph of searching (Fig. 15a). It can be said that we have programmed the most general model of the outside world. In solving a concrete problem it can turn out that the graph defined by the initial image Π_0 does not correspond to this problem. For example, the pictures in the training collection may not have "holes" ($FP_0 = P_0$), and the search would have to be done by a reduced graph (Fig. 15b). The program may find this out only experimentally, by applying the operator *contour filling* (*F*) to the collection of training pictures (P_0). However, having seen that the collection of pictures obtained at the output of this operator coincides with the input collection, we can, on the basis of experimentally obtained information, change Π_0 accordingly and, consequently, change the graph of searching/execution. Thus in the process of working the program is able to find out certain details about the concrete situation with which it is confronted. On the basis of these details it changes (complements) the model it has been given.

Here we do not describe in detail the machine realization of the scheme, roughly described above, of using the model for reducing the search, since, on the one side, this realization embraced a specific (and not very large) set of drawing operators and operators of dividing into fragments and, on the other side, it has used a number of specific procedures, related to the particular features of the computer's command system.

4.3. Order of searching. 4.3.1. Searching of operator and searching of input collections. As it was already mentioned, a single step of program's work consists in that one of the elementary operators processes one (or several, if it has several inputs) of the collections found in memory and then saves its output data.

The question is, which of the operators should be applied at a given moment, and to which of the collections of objects they should be applied. In the program this choice is done in two steps. First, by the scheme described in §4.3.4, the program chooses the subsequent operator and puts it to work. This operator, in correspondence with the procedure described in §4.3.3, chooses from memory an input collection of objects and processes it.

4.3.2. Storing of objects in memory. Collections of objects are numbered in the same order in which they are saved. The numeration is done separately for each of the types of objects and for each of the levels. Not all the collections of Boolean numbers (the results of work of drawing operators, the *threshold operator*, the operator *comparison*, and the *logical operator*) are stored in memory. Trivial (containing only zeros) collections of Boolean numbers, as well as collections identical to the ones already kept in memory, are not stored.

All the collections of numbers (obtained at the output of measuring operators and the operator *number of fragments*) are stored in memory. In general, such a rule of storing objects in memory may lead to creation of loops. Indeed, if in a certain system there exists an operator O and a collection of numbers N such that $N = ON$ then this method of storing the operator O will fill the memory with identical collections of objects N. However, it can be shown that in our system of operators this cannot happen and, therefore, such a method of storing is acceptable. Moreover, it is useful, because the probability that there appears a collection of numbers completely identical (including the dimension) to one of the collections stored in memory is very small. Consequently, checking for identity would, on the average, take much more time than the possible cases of processing identical collections of numbers.

To keep in memory all the collections of pictures is an impossible task. The only collection stored is that of training pictures; as for the other collections, only the method of obtaining them (in the form of a sequence of operators) is stored. If need arises, a collection of pictures is generated again from the collection of the training pictures. This allows us to get by using a small memory capacity, even though it considerably increases the working time of the program, especially for the problems where it is necessary to build many levels and many unions.

4.3.3. Retrieving objects from memory. Let z_0 be the total number of objects (of a certain type, on a certain level) and z the number of collections of objects already processed by the given operator. When the operator starts working, it takes out of memory the next z+1 collection of objects, processes it, and stores in memory the output collections — if there are any.

This general scheme requires a more precise definition, since 1) operators can extract objects from different levels and 2) some operators have several inputs, and this requires a more complicated organization of accounting for the already processed collections of numbers.

If there exist not yet processed collections of objects on several levels at once, the operator takes the next collection from the top level (with the smallest index). The inputs of the operator, if there are more than one, are ordered. The input order corresponds to the order of their appearance in the corresponding column of the list of operators on page 8. When such an operator receives the control, it recovers from memory not just a pair (or a triplet — according to the number of inputs) of collections of objects, but several such pairs. For one input of the operator, only one collection of objects gets extracted from memory (one of the two elements of such pairs); for other inputs the entire sequence of collections of objects gets extracted. Then the operator processes all these pairs and stores

in memory the collections obtained at the output. In this way, one step of operator's work consists in processing all the pairs of collections of objects taken from memory.

Let us look now at the rule of recovering from memory the input collections of objects for operators having more than one input. For simplicity let us suppose that we have an operator with two inputs (for example, the *threshold operator*) and z_0' is the total number of collections of numbers, z_0'' is the total number of collections of Boolean numbers in memory, and z' and z'' respectively are the quantities of "already processed" collections of numbers and of Boolean numbers. This situation is shown in Fig. 16 where the pairs of the input collections of objects are represented by outlined points. The pairs of collections already processed by the given operator are represented by black points. Two situations are possible: 1) if $z' < z_0'$, then for the first input the next $(z' + 1)$ collection of numbers is taken from memory, for the second input all the collections of Boolean numbers with the indices from 1 to z'' are taken; 2) if $z' = z_0'$ and $z'' < z_0''$, then for the first input all the collections of numbers from 1 to z' are taken from memory and for the second input the next $(z'' + 1)$ collection of Boolean numbers is taken.

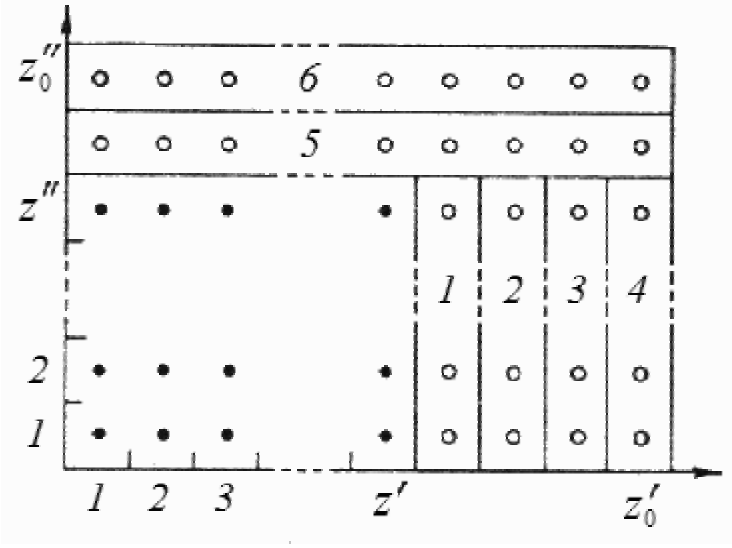


Figure 16. The order of recovering from memory the pairs of collections of objects that go to the inputs of the operators with two inputs.

This rule of choosing objects from memory can be illustrated by the following example. Let it be that in the situation shown in Fig. 16 the control is given over to the same (*threshold*) operator several times in a row. Then the first time this operator will recover from memory and subsequently process z'' pairs of collections of objects (in Fig. 16 these pairs occupy column 1). One element of all these pairs will be the same $(z' + 1)$ collection of numbers, and the other element will be one of the collections of Boolean numbers (from 1 to z''). After this the *threshold operator* will increase the value of z' by 1. When the *threshold operator* will be in control next time, it will again process in one step z'' pairs of input collections of objects (column 2 in Fig. 16). After the fourth application of the *threshold operator* (if meanwhile the quantity of collections of numbers in memory hasn't increased and z_0' hasn't changed) it turns out that $z' = z_0'$; consequently, if the *threshold operator* will be in control for the fifth time, it will process z_0' pairs (occupying line 5 in Fig. 16), etc.

4.3.4. The order of giving the control over to elementary operators. It is convenient to analyze the work of Block (I), which organizes construction of new levels (this block directs the work of the operator *division into fragments*) and Block (II), which directs the work of all other operators on the levels already built.

The second block contains the operators which are applied only to pictures (list A on page 8) and the operators processing the collections of numbers and Boolean numbers (list B on page 8). The order of work of the operators of Block II is defined by the ordered list of operators (the general sequence of operators in lists A and B). At each step of its work the block gives the control to the first operator on the list that still has some unprocessed collections. The second block works until all the operators will have processed all the input collections of objects (including the new ones, constructed by the operators of this block).

The first block controls the operators dividing the pictures into fragments. Like the operators of the second block, these operators take stock of the input collections of objects — collections of pictures they have already processed.

If we don't take into account unimportant details, it can be said that the entire search scheme of the program works according to the single, ordered list of operators (p. 8). In the beginning there exists only one unprocessed collection of objects — the collection of training pictures. According to the ordered list, the measuring operators are applied first. Their output collections of numbers are stored in memory. Then the first of the drawing operators starts working. As a result of its work, a new collection of pictures appears in memory; consequently, the control again passes over to the measuring operators, etc.

There exist in the program the following restrictions on this general scheme of search: 1) the operator *union* is applied only to pictures obtained as a result of work of the operator *division into fragments*, and 2) the operator *division into fragments* is not applied to the pictures obtained as a result of union.

The program finishes its work if 1) in the process of working of the second block, the *decision making operator* finds a collection of Boolean numbers of the first level, correctly dividing the pictures into classes or 2) the block of building new classes exhausts all the methods of obtaining new pictures, that is, the program did not find the distinguishing rule.

Unfortunately, while creating an algorithm of searching of operations it was not always possible to guide ourselves only by the considerations of usefulness of this or that order of search. The small volume of operational memory of the machine imposes serious limitations on the organization of the searching of operations. For example, the order of search can be arbitrarily changed inside of each of the lists A and B, but it is not possible to transfer operators from one list to the other. Such an organization of searching results in that each operator from list B can be applied only after the completion of all the re-drawings and measurements defined by list A.

5. Experiments in training the system

5.1. The machine implementation of the system. 5.1.1. The program. The present variant of the program has been created for the computers M-20 and БЭСМ–3М with a memory volume of 4000 words. The program takes up more than 3000 words. In the

process of creating the program an additional "memory block" of 4000 words was connected to the machine. In general, all the program's work was done by one "block", the second one being used only for work fields — the outputs of the operator *union*.

Since the program contains mostly logical operations, the speed of the machine approached 40,000 operations per second. As shown by experiments, such a speed proved quite sufficient. The machine spent very little time on solving any of proposed problems (as compared to the typical calculation problems). However, the small volume of the operational memory of the machine was affecting considerably the totality of problems being solved. The restrictions were produced by the necessity of keeping in memory all the obtained collections of numbers and Boolean numbers. The total volume of memory used to keep these collections equaled 1000 words, and the maximal acceptable number of collections of numbers and, separately, of collections of Boolean numbers, was 128 on each level.

Such an "overcrowding of memory" is possible mostly in those problems where the program (with the help of various operators of division into fragments) is able to build many levels of objects. In such cases, unions of different fragments of the pictures with the subsequent re-drawings and measurements created a large quantity of collections of numbers. In their turn, the *logical operator* and, especially, the operator *comparison* created too many collections of Boolean numbers. In order to avoid this, in solving the majority of the problems we used a simplified version of the system. In the corresponding list of elementary operators the following operators were omitted: *separation by boundaries, separation of lines by branching nodes* and *comparison*. Only for those problems where it was necessary for the description of the distinguishing rule, the corresponding operators were added to the abridged list. In this case, the solution process could be completed only for relatively simple problems.

5.1.2. The training material. The pictures were first coded in the form of a 45 × 64 binary matrix and put onto punched cards. From these a collection of training pictures — a problem — was composed. In addition to the pictures themselves $p_0^i \in P_0$ the following information was inputted: 1) the number of the problem, 2) the number of pictures in the problem, 3) two collections of Boolean numbers, defining the separation of the collection of training pictures into two classes.

$$b_+^i = \begin{cases} 1, & \textit{if picture } p_0^i \in 1^{st} \textit{ class} \\ 0, & \textit{in the contrary case} \end{cases}$$

$$b_-^i = \begin{cases} 1, & \textit{if picture } p_0^i \in 2^{nd} \textit{ class} \\ 0, & \textit{in the contrary case} \end{cases}$$

Since the same picture cannot belong simultaneously to both the first and the second classes, for the same *i* it cannot be at the same time that $b_+^i = 1$ and $b_-^i = 1$. However, such pictures p_0^i are possible for which it is not indicated to which class they belong: $b_+^i = b_-^i = 0$.

The majority of “interesting” problems can be formulated using a small number of pictures; thus most of the problems contained 4 + 4 pictures; sometimes the total number of pictures in the problems came up to 16. In the experiments with the system we used altogether 48 problems (see the *Appendix*). The problems are separated into two classes (the left and the right). The Appendix also contains answers — the description of classes or the principle of classification in “human” language.

5.1.3. The process of searching for a solution. The task of the program was to find the distinguishing rule that would differentiate each picture of the left class from the picture of the right class. Since each distinguishing rule is a superposition of a certain number of elementary operators, and the work of each of the elementary operators can be quite concisely described in “natural human language”, a person can easily predict how the program would classify, according to this distinguishing rule, some new pictures not presented to it in the training. Thus, it is the behavior of the program while searching the distinguishing rule and not while being tested, which interests us. The variant of the program described here did not contain the block “test” and had as its single working mode the search for the distinguishing rule (“training”).

For such a natural and relatively simple collection of elementary operators a person can also predict, to a certain extent, the behavior of the system in the training process. This is easy to do for simple problems characterized by a certain monotony of pictures and, correspondingly, by a small number of parameters. However, the experiments have shown that a large number of the possible search branches, which appear in solving more complicated problems, usually pass unnoticed. Moreover, some disparities between the behavior of the system and that of a person may be found only in experiments with the working program.

In the process of searching for the solution the program prints out the record of training proceedings. For each new collection of objects appearing in memory its “family tree” is printed out. The program says with the help of which of the operators and from which of the input collections of objects the given collection has been obtained. When the decision operator finds on the first level a collection of Boolean numbers correctly dividing the pictures of the training collection into classes, the index-number of this dividing collection is printed out, and the machine stops. After this the record of proceedings allows us to find the sequence of operators defining the principle of classification.

There exist two natural indicators characterizing the work of the program in solving a problem: 1) the solution time and 2) the number of intermediate results (the volume of memory used up at the moment of finding solution). The results of the work of the program containing a reduced number of elementary operators are shown in Table 2. For each problem are given: the solution time, the number of constructed levels, and the number of intermediate results; the quantities of collections of numbers and of collections of Boolean numbers are given separately.

Table 2

Problem number	Level	Solution time	Quantity of collections of numbers	Quantity of collections of Boolean numbers	General number of distinguishing rules
1	1	15 sec	9	2	4
2	1	20 sec	15	6	4
3	2	1 min	37	20	8
4	2	3 min 25 sec	102	65	4
5	1	25 sec	21	16	9
6	2	35 sec	31	6	4
7	2	40 sec	31	14	4
8	2	2 min	75	6	6
9	2	3 min 30 sec	75	4	6
10	1	35 sec	21	10	60
11	2	50 sec	25	8	8
12	2	45 sec	51	24	14
16	3	4 min 15 sec	218	61	4
17	2	1 min 50 sec	43	8	18
18	2	2 min 15 sec	62	13	4
19	2	3 min 55 sec	261	16	6
20	2	4 min 25 sec	236	53	5
21	2	3 min	140	26	5
22	1	20 sec	21	8	2
23	1	20 sec	15	2	2
24	2	2 min 45 sec	83	12	2
25	1	25 sec	9	6	1
26	1	20 sec	21	8	2
27	2	35 sec	37	9	24
28	2	1 min 10 sec	43	37	6
29	2	2 min 15 sec	69	12	8
30	3	5 min 35 sec	327	73	2
31	3	4 min 40 sec	179	49	4
32	2	2 min 45 sec	89	14	2
33	2	2 min 20 sec	57	10	2

5.2. The number of distinguishing rules in a problem. 5.2.1. Distinguishing rules chosen by the program. We call *solution of a problem* the distinguishing rule (among all possible ones) that is found first during an experiment. Distinguishing rules for each of the problems are described in §5.5. However, the solution itself does not yet give a complete characterization of the program's work. It is interesting to know which different principles of classification the program can find for a concrete problem. In order to do that, we have conducted special experiments, in which the program, once the solution had been found, did not stop working but erased the corresponding collection of Boolean numbers and continued to look for the next one. It turned out that in many problems the program found not just one, but several — sometimes many — distinguishing rules.

Table 2 shows the number of distinguishing rules found for each problem. Regrettably, the limited memory capacity did not always allow us to bring the experiment to the end, that is, to do a complete search for all the possibilities. Thus, for some problems we show only a certain lower bound of the complete number of distinguishing rules being chosen.

Usually, for each problem there are many distinguishing rules formulated in the language of elementary operators. Naturally, most of them are superstitions, that is, are formulated only on the basis of the concrete training material but do not correspond to the distinguishing rules proposed for the given problem by a person. All the other distinguishing rules can be called *synonyms*, since they give identical divisions for the entire set of all the possible pictures of the given problem.

5.2.2. Synonymous distinguishing rules. In most cases, problems have a certain symmetry of classes, so the program can formulate the description of each class separately. As a rule, distinguishing collections of Boolean numbers, corresponding to such synonymous principles of classification, appear simultaneously at different outputs of a certain operator (a drawing operator, *logical operator*, or, sometimes, *threshold operator*) and thus in the record of the solution the problems are situated next to each other. Hence, the first pair of distinguishing rules in Problem #1 results from applying the operator *contour isolation*, one of the Boolean outputs of which corresponds to the classification "the right class — contour pictures", and the other, to the classification "the left class — non-contour pictures".

Often the presence of several distinguishing rules can be explained by the fact that the same characteristic, relevant for the distinguishing rule, can enter in its formulation in different ways. The most typical example is Problem #29. The important characteristic here is the position (x-coordinate) of an outside point. This coordinate is broken up into two lots: left points and right points. Each one of the two collections of Boolean numbers resulting from this break-up can enter into the formulation of the distinguishing rule. The application of the *logical operator* to the collection of Boolean numbers characterizing the left outside points gives the following classification principles:

1. Left class: pictures having an isolated point on the left part of the raster,
2. Right class: pictures not having an isolated point on the left part of the raster.

The same collection of Boolean numbers makes it possible to construct another description of the distinguishing rule, using the operator *number of fragments* and the following separation by threshold. Indeed, the number of isolated points in the left part of

each picture is either 0 or 1 and therefore can be broken up into two lots. Hence two more classification principles:

3. Right class: pictures in which the number of outside points on the left part of the raster equals 0,
4. Left class: pictures in which the number of outside points on the left part of the raster equals 1.

Analogously, four more distinguishing rules can be constructed, using another collection of Boolean numbers, characterizing the outside points on the right part of the raster. In this way, only the use of the *x*-coordinate of an isolated point gave us the whole cluster of eight distinguishing rules. A similar cluster of distinguishing rules (all of it or just some part) can be seen in many other problems. Such are, for example problems #3, #4, #17, #31.

5.2.3. Superstitions. One of the causes of superstitions — distinguishing rules that are unnatural from the human point of view — is the possible discrepancy between the language of elementary operators and the language used in such cases by people. In the commentaries to Problems #23, #24, and #28 we discuss the entire class of distinguishing rules that can be considered superstitions. They appeared due to a certain inadequacy of the language of elementary operators we have chosen. The second cause consists in that any sufficiently rich language in principle admits the construction of a large number of too complicated distinguishing rules.

What does the program do against this variety of distinguishing rules?

First, as it has already been mentioned, the majority of the distinguishing rules that are superstitions are eliminated thanks to the procedure of breaking up into lots during the selection of thresholds.

Secondly, thanks to the special characteristics of the storing procedure, the program eliminates the superfluous branches of search and thus reduces the number of distinguishing rules being generated. The collection of Boolean numbers obtained for the second time — and thus in a more complicated way — is not stored in memory and therefore the correspondent complicated distinguishing rule is not constructed. Analogously, thanks to the organization of search of re-drawings and breaking up into fragments (see §4.2), only the shortest of all possible descriptions of distinguishing rules are chosen.

Let us analyze, for example, Problem #1. The simplest distinguishing rule here consists of a single operator, that of *contour isolation*. The Boolean outputs of this operator correspond to the statement “to the right class belong the contour pictures, to the left class, the non-contour ones”. The program chooses this distinguishing rule for Problem #1. However, another — more complicated — operator also corresponds to this training collection: first fill the contour (this will not change the pictures of the training collection) and then apply the operator *contour isolation*. One more variant of a more complicated principle of classification, “to the right class belong the pictures all of whose parts are contour” (first break the pictures up by connectedness, then isolate contour of the obtained fragments and apply the *decision making operator* to any of the Boolean

outputs. Even though both rules satisfy the condition of Problem #1, the program does not construct them since they contain superfluous operations.

In the problems that were given to the program (See *Appendix* and Section 5.5), the superstitions were never first to be formed, but the possibility of their appearance (in some problems) cannot be excluded. As a rule, in simple problems with a small number of parameters stored in memory, superstitions are impossible simply because there is not enough material to construct them. Such are Problems #1, #8, #25, #27. However, in problems with a large number of irrelevant parameters superstitions can be built. Usually, as it has been said, they appear at the later stages of searching for a solution. As an example of a superstition we will give here the description of one of such distinguishing rules. In Problem #28 the classes differ by the position (*x*-coordinate) of the white figures in the picture. The program first divides the pictures into fragments. The application of the operator *contour isolation* distinguishes between contour and not-contour fragments, and the application of the operator *contour filling* separates the fragments into full and empty (notice that in this problem the contour fragments are not necessarily empty and the empty fragments are not necessarily contour). The breaking up into lots of the *x*-coordinates of only the contour fragments results in the following classification principles:

1. To the right class belong the pictures that have a contour part on the right;
2. To the left class belong the pictures that do not have a contour part on the right.

Analogously, two more distinguishing rules can be obtained by breaking up into lots the *x*-coordinates of the subset of empty fragments (defined by the Boolean output of the drawing operator *contour filling*).

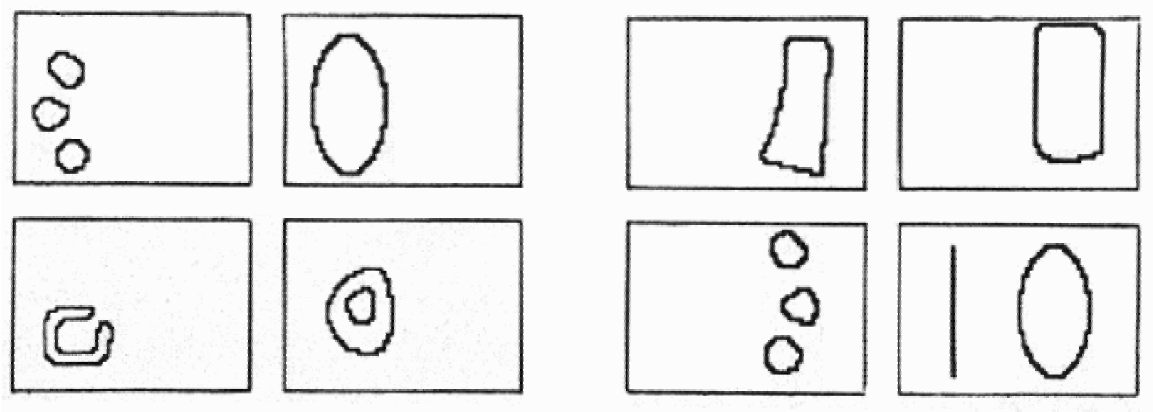


Figure 17. Union of the contour fragments of Problem #28.

Superstitions appear later. After the application of the operator *union* to the subset of contour fragments the pictures of training collection turn into the collection of pictures shown on Fig. 17. It has to be noted that the *x*-coordinates of these pictures cannot be broken up into lots. Apparently the center of gravity of the bottom right picture of Fig. 17 lies somewhere in the middle between the two lots and hinders the break-up. However, if we apply to these pictures the operator *contour filling*, the center of gravity of this picture moves to the left. The *x*-coordinate of such filled pictures breaks up into two lots. All four pictures of the left class come into one lot, while the pictures of the right class come into

the other lot. It can be said that such a cumbersome distinguishing rule coincides, to a certain extent, with the true principle of classification. However, people probably would not formulate the distinguishing rule of Problem #28 in such a complicated way. Therefore it must be called a superstition.

5.3. Effect of expanding the training collection. What does such expanding lead to?

Two cases are possible here. The first case is when the added pictures contradict the classification principle found for the first problem. Then, naturally, the program will find some new principle of classification. It will, of course, satisfy also the initial problem, but, in the absence of additional pictures, the choice of a more cumbersome classification principle would be excluded. The sequence of Problems #1, #2, #3 illustrates this case.

The other case is when the new material does not contradict the classification principle found in the first problem. It may seem that such an expansion of training material is senseless because it apparently only makes the training time longer, as more pictures have to be processed. However, as we have already remarked, the scheme of picture processing is not a fixed one but depends on the material given to the program. The time of solving a given problem is defined not only by the size of the training collection and the complexity of the distinguishing rule, but also by the number of dead-end branches of searching that the program has to go through before finding this rule. Such dead-end branches may appear in the problem for example, when some superfluous parameters get broken up into lots. Such an accidental break-up into lots may not yet lead to superstitions (provided it is not used in constructing the distinguishing rule), but the search of all the consequences of such a break-up will take some time.

The existence of such superfluous break-ups into lots tells us that the choice of training material has not been altogether successful. A change of training material (in particular, its expansion) can make these spontaneous break-ups into lots disappear, and the dead-end branches will be eliminated much earlier. This will reduce the search, and in some cases the time saved from such a reduction may even compensate the time spent on processing the additional pictures. For example, in Problem #21 the 1.5 times increase of the training material in comparison with the Problem #20 reduces the training time approximately 1.5 times. It can be said that the additional training material makes it easier for the program to find the solution. Other examples of speeding up the solution process as a result of increasing the training material are the pairs of Problems #32–33, and #47–48.

5.4. Training with the use of test material. In the training regimen the program was given a collection of pictures divided into two classes, and it had to find a classification principle satisfying this division. The program did not have the test regimen, that is, the classification, on the basis of the already chosen distinguishing rule, of new (test) pictures as belonging to one of the classes. However, to a certain extent the test could be conducted during training sessions. For this, the collection of training pictures had to contain not only examples of pictures of each class, but also test pictures.

As mentioned in §5.1, the information about which picture belongs to which class was recorded in special information cells of the training collection. If the training collection also contained test pictures, the machine simply was not told to which class each of them belonged. Therefore, while choosing the distinguishing rule (with the help of *decision*

making operator), the program paid attention only to examples of pictures of each class. Of course, this does not mean that the addition of test pictures did not affect the training process. The test material could contain information, relevant for constructing the distinguishing rule. Such a possibility of extracting information from the test material was analyzed already in M Bongard's book [5].

In the simplest case such addition of test material may not affect the training process and thus not differ from a separately conducted test. In the collection of Boolean numbers obtained as a result of training, 1 will correspond to each picture of one class, and 0, to each pictures of the other class. The values put in this collection into correspondence to the test pictures will say to us into which class the program put each one of these pictures. Here such an experiment doesn't give anything except a simple corroboration of the fact that the application of elementary operators in order corresponding to the chosen distinguishing rule indeed allows us to classify the test pictures. (See the solution of Problem #24 in §5.5)

Problem #44 differs from Problem #43 by four additional test pictures. As a result of training on this problem the program chose the same distinguishing rule as in Problem #43. Two test pictures on the left are said to belong to the left class, two pictures on the right, to the right class. The training time increased in correspondence with the increase of the training material.

In solving Problem #8 the program broke into lots the number of fragments in the picture: it can equally be 6 or 7. This parameter turned out to be irrelevant for the distinguishing rule. The addition of six test pictures to this problem (Problem #39) showed that there exist pictures for which the number of fragments is different from 6 and 7. Thus in solving Problem #9 the program does not break up into lots the number of fragments anymore and, as the Table shows, the stored Boolean numbers in this case are fewer than in Problem #8. In this example the break-up into lots of a superfluous parameter (number of fragments) practically did not affect the ensuing search. Therefore, the information, extracted from the test material in Problem #9, which has permitted the elimination of the irrelevant parameter, did not influence the working time of the program: the increase of training time in Problem #9 approximately corresponds to the increase of the input collection of pictures. Of course, the addition of training material (see §5.3) can sometimes speed up the search and so facilitate the finding of the solution.

The clearest example of extracting information from the test material is given by the problems in which the training material is so small that it is difficult to choose a single classification principle from many principles of similar degree of complexity. In what do the pictures in Fig. 18 differ? We can say that the pictures differ by the number of figures, or their size, or that the figures in the picture on the right are spread across a larger area in comparison with the picture on the left, etc. Which of these classification principles shall we choose? The solution can depend on the kind of pictures we are to classify on the test. If in the test there is a wide spectrum of sizes (and quantity) of the individual parts of the image, but the pictures differ clearly by the size of the area occupied by the image (see Problem #10), then it will be the area of the figure obtained as a result of applying the operator *convex hull* to each of the pictures that will be broken up into lots during the training. In such case it is precisely this type of classification that will be chosen as the distinguishing rule.

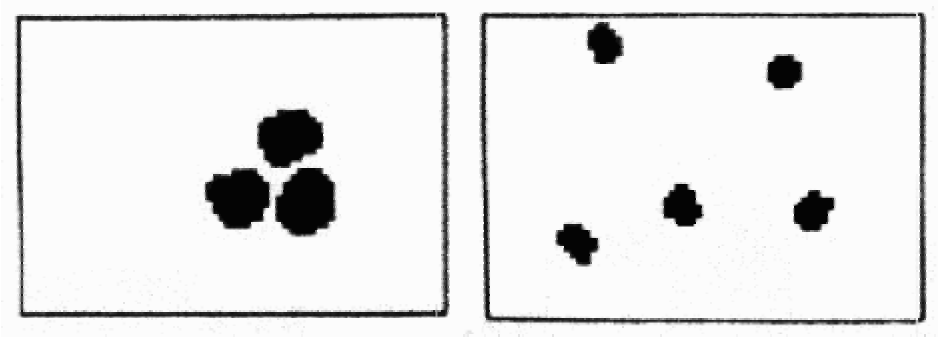


Figure 18. Two training pictures.

If we choose different material for the test, with the same examples of pictures of each class (see Problems #11 and #12), then the program will choose other distinguishing rules.

5.5. Commentaries to the problems. ***Problem #1***. The capacity of the drawing operator to give at the output not only collections of pictures, but also collections of Boolean numbers gives us the possibility to build distinguishing rules (based on such characteristics as "contour", "convex", etc.) also without applying measuring operators and the *threshold operator*.

Problem #1. is the simplest of all problems given to the program. Application of the operator *contour isolation* gives at the output, in addition to contour pictures, also two collections of Boolean numbers which define, correspondingly, the subsets of "contour" (segments of straight lines) and "non-contour" (ellipses) pictures. The *decision-making operator* finds out that the very first one of these collections of Boolean numbers correctly divides the pictures into classes. Such is the principle of classification. However, the program spent much more time searching for it, than if would have spent simply processing in this way the eight pictures. As we noted in §4, the sequence of execution of operations in the program (defined by the order of operators in the lists A and B on p. 37) is such that the program proceeds with re-drawings only after having completed all the measurements of pictures; and after having measured all the pictures obtained as a result of all re-drawings, the program will, finally, address the *decision making operator* which will find the distinguishing collection of Boolean numbers. It is on these useless re-drawings and measurements that the program spends most of the time.

Problem #2. We have added two contour ellipses to the pictures of the left class, so that these classes will not be distinguishable by the characteristic "contour" – "non-contour". The distinguishing rule, found by the program, is defined by the sequence of operators: 1) *contour filling*, 2) apply the operator *contour isolation* to the output. The Boolean numbers at the output of this operator divide the pictures into classes. These Boolean numbers can be interpreted as the truth values of the statement "in the picture a figure is shown that stays contour even after the contour filling".

Problem #3. Further complication of the same classification principle. Here the program passes to process the individual parts of the image, since the general (integral) characteristics of pictures give no solution. In order to build the distinguishing rule the

program has to divide the pictures into fragments by connectedness and then apply to the obtained fragments the same sequence of operations as in Problem #2.

After this, the *logical operator*, being applied to the collection of Boolean numbers characterizing the straight-line segments, builds on the first level a collection of Boolean numbers corresponding to the statement “in the pictures of the first class there exists at least one part that stays contour even after contour filling”.

The resulting distinguishing rule is the following: divide the input picture by connectedness, then apply the operator *contour filling* to the pictures-fragments obtained by the previous operation; if, as a result of the last operation, there has been obtained at least one contour picture-fragment then the input picture belongs to the right class.

Problem #4. Unlike in the previous problem, here even the pictures of the left class can contain the right line segments. The preliminary contour filling reduces this problem to the previous one. The distinguishing rule still does not use measurements and looks like this: apply to the input picture the operator *contour filling* and divide the result by connectedness; if as a result of the last operation there has been obtained at least one contour picture-fragment, then the input picture belongs to the right class.

Problem #5. This is the simplest variant of the problem in which the distinguishing rule uses measurements with the subsequent cutting by threshold. In this problem the slope of the figures (Fig. 10) is broken up into lots. The solution of this problem requires a bit more time than that of Problem #2, which also contains 12 pictures, because the re-drawings and measurements of such pictures take up more time.

Problem #6. In this problem it is the thickness (the size of the small axis) that is broken up into lots. In order to find this out the program must first divide the pictures into parts and measure the thickness of each of the fragments. Then application of *threshold operator* gives at the output two collections of Boolean numbers defining the subsets of “thin” and “thick” parts. After this, application of the *logical operator* to the first of these subsets gives on the first level the distinguishing collection of Boolean numbers corresponding to the statement “in each picture of the left class there exists a thin part”.

Problem #7. This problem differs from the previous one not only in that there is another parameter in the distinguishing rule — not the thickness but the length (the size of the big axis) of the figure — but also in that here an intermediate re-drawing is necessary. The program measures and breaks into lots the size of the axes after applying operator *contour filling* to the parts of the pictures.

Problem #8. An example of using the operator *union*. A part of the searching tree that the program follows while looking for the distinguishing rule is shown in Fig. 19. The program first divides the pictures into fragments, then unites separately the black and the white parts, applies various re-drawings to them (in particular, *convex hull filling*) and finds out that the slope of the convex hull over the union of the white (contour) figures can be broken up into two lots corresponding to the vertical and horizontal orientation. The resulting collection of Boolean numbers gives us the distinguishing rule. The solution time here is somewhat prolonged because the program first makes an unsuccessful move — tries to unite the non-contour (black) figures.

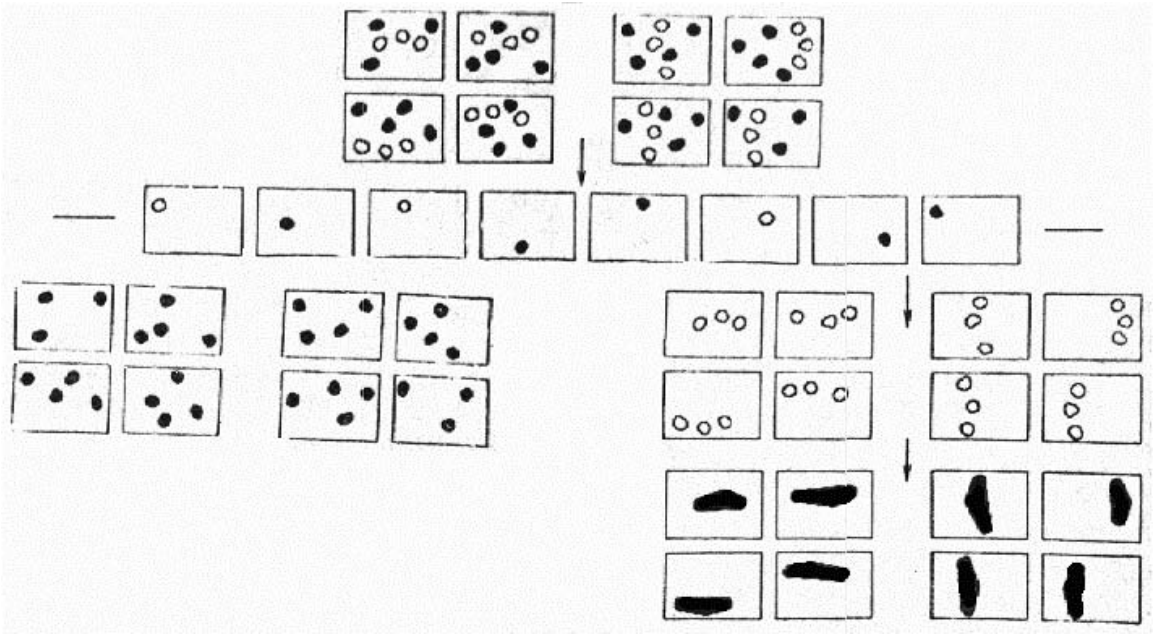


Figure 19. The process of finding a solution for Problem #8.

Problem #9. Training with the use of test material.

The distinguishing rule is the same as in Problem #8: to divide the input picture by connectedness; apply to the picture-fragments the operator *contour isolation*, one of the Boolean outputs of which defines the subset of the contour parts of the picture; to the subset of the contour fragments apply the operator *union* and measure the slope of the figure obtained; if it is oriented horizontally, the picture belongs to the right class; in the contrary case it belongs to the left class.

In correspondence with this rule the program put in each class three pictures of the test material.

Problem #10. Training on the test material.

Since in the test material there exists only one example of picture of each class we can regard as the distinguishing collection any collection of Boolean numbers in which 1 will correspond to a picture of one class and 0 to a picture of the other class, no matter what values this collection of Boolean numbers will take in other pictures. The first collection of distinguishing Boolean numbers was obtained as a result of the following sequence of operations: operator *convex hull filling* was applied to the pictures and the area of the convex hull was measured; the last parameter was broken up into two lots. The picture of the left class got into one lot, the picture of the right class, into the other. Correspondingly, the test pictures were grouped into classes according to whether the area occupied by the image is small (left halt of the test material) or big.

Problem #11. Training on the test material.

The program found the collection of Boolean numbers distinguishing the picture of the left class from that of the right class only after it had divided the pictures by connectedness. The area of fragments was broken up into two lots and the *logical operator* (using the collection of Boolean numbers corresponding to the small fragments)

supplied four collections of Boolean numbers to the first level at once. These correspond to the statements: a) “all the fragments in the picture are small,” b) “not all the fragments in the picture are small,” c) “there is no small fragment in the picture,” d) “there is a small fragment in the picture.” After this, the *decision-making operator* finds out that all these collections of Boolean numbers are distinguishing.

It has to be noted that, in correspondence with these classification principles, the bottom left picture of the test material falls sometimes into the right and sometimes into the left class. Thus it can be said that the program is not able to unequivocally decide to which class this picture belongs — and neither are people.

Problem # 12. Training on the test material.

The distinguishing rule: divide the input picture by connectedness, then count the number of fragments. The picture belongs to the left class if it contains three fragments.

Problem # 13. The distinguishing rule: divide the input pictures by boundaries, then count the number of fragments. Pictures belong to the left class if they contain three fragments; otherwise they belong to the right class.

Problem # 14. Unlike in the previous problem, here we must divide the pictures by both connectedness and boundaries. Application of the operator *contour isolation* to the obtained fragments will reveal subsets (Boolean outputs of this operator) of white and black rectangles. Subsequent application of the operator *number of fragments* to the collection of Boolean numbers corresponding to the black rectangles gives us the collection of numbers on the first level. This collection is broken up into two lots — the number of black fragments may equal 1 or 3. This is why application of *threshold operator* to this collection of numbers gives at the output two collections of Boolean numbers, each one of which is distinguishing.

Problem # 15. This problem is somewhat more complicated. What is relevant for the distinguishing rule here is not the total number of black rectangles in the picture, but their number in the individual fragments obtained via division by connectedness. Application of the operator *number of fragments* to the collection of Boolean numbers of the third level, characterizing the black rectangles, gives on the second level a collection of numbers which is broken up into three lots — the number of black rectangles in each fragment of the picture may equal 0, 1, or 2. After this, application of the *logical operator* to the collection of Boolean numbers corresponding to the third lot will give on the first level two collections of Boolean numbers, corresponding to the statements “in the picture there exists a fragment which contains two black rectangles” and “in the picture there doesn’t exist a fragment which contains two black rectangles.” Both of these collections of Boolean numbers will be distinguishing.

Problem # 16. The distinguishing rule: divide the pictures by connectedness; apply the operator *contour isolation* to the obtained fragments; divide the result into fragments by connectedness; if at least one picture will be divided into parts (more than one), then the picture belongs to the left class.

Problem # 17. This problem would have been very simple and would have been solved in a few seconds if it weren’t for the small-point “noise.” In order to get rid of it, we have to divide the pictures into fragments. Almost two minutes of machine time is spent on

different re-drawings and measurements of more than a hundred of pictures-fragments, which depict mostly just one black point. As a result the program finds the distinguishing rule in the form of the following chain of operators: divide the input picture by connectedness, apply to the pictures-fragments (obtained as a result of the previous operation) the operator *contour filling*; if as a result of the previous operation we get at least one concave picture-fragment then the input picture belongs to the left class.

It may seem that such a division into fragments by the general scheme is not the most efficient method of fighting the "noise." Application of some regular method of getting rid of small points would eliminate the meaningless re-drawings and measurements and considerably shorten the solution time. In reality, however, the long time of training is not a result of shortcomings in the program's construction, but the price we had to pay for its universality. Thanks to this trait the program can solve, in particular, such problems where it is precisely the small points that are relevant for the distinguishing rule. It is not possible to tell beforehand which detail of the image is "noise", that is, irrelevant for classification. This situation is illustrated by the following problem.

Problem # 18. The same collection of pictures as in Problem #17. After the division of pictures into fragments, the operator *union* is applied to continuous (non-contour? connected?) fragments (the subsets of continuous fragments are defined by the collection of Boolean numbers appearing at the output of the operator *contour filling*), that is, individual small points. As a result we obtain the same pictures, but without big contour figures. Applying to them the operator *convex hull* gives us massive elongated figures. Then successive application of the operator *slope of the figure* and *threshold operator* gives two collections of Boolean numbers defining separately the subsets of pictures with the figures, elongated approximately horizontally, and the figures, elongated in the vertical direction.

Problem # 19 In order to find out that in the pictures of the right class the small points are situated outside the closed lines, the program must first apply to each picture the *operator contour filling.* As a result all the internal points will merge with the background of the filling. Then it is necessary to divide the obtained pictures into fragments and isolate the subset of small points (the points left after contour filling, unlike all the other parts of the pictures, are "contour figures"). Finally, application of the *logical operator* to the collection of Boolean numbers characterizing these points gives us the distinguishing collection of Boolean numbers on the first level.

The solution is the same distinguishing rule as in Problem #4.

Problem # 20 Just as in Problems #4 and #19, here all the internal parts of the pictures only serve to "distract attention". The classification of the picture is attained by contour filling, division of the obtained picture into fragments and measurement of the area of these fragments (which in the training material is broken up into two lots). Any picture, which, after contour filling, contains at least one small compound fragment, belongs to the left class.

The search for distinguishing rules is done in such a way (see §4) that in this problem the program spends much time analyzing the "diversion" details of the pictures. First the program builds the second level — divides the pictures into fragments. Then, after isolating the contour, it constructs the third level via repeated division by connectedness.

Here it turns out that the fragments break up into three lots by their size. The smallest parts in their turn break up into vertical and horizontal. All these characteristics are irrelevant for the division. However, their processing and the application of the operation *union* to the obtained subsets of fragments takes much time. Only after this the program finally starts checking that branch of the searching which leads to the distinguishing rule: first it fills the contour and then divides the obtained pictures into fragments.

Problem # 21. Four supplementary pictures were added to the training collection in order to get rid of the multitude of irrelevant characteristics of pictures — at least partially. Because of the size-variations of the inner fragments of pictures, the breaking up by this parameter does not happen anymore. As a result the solution was arrived at quicker, the increase of the number of pictures in the training collection notwithstanding.

Problem # 22. Measurement of the length of lines and the subsequent separation by threshold (whose value is chosen by the procedure of breaking up into lots) divide the pictures of the training collection into classes.

Problem # 23. In the program the measuring operator *length of lines* may be applied to any contour pictures. Consequently, in principle there may be quite long distinguishing rules, ending by contour isolation, measurement of its length and separation of this length by threshold. At first sight such distinguishing rules seem quite natural. However, the experiments have shown that at this point there is a certain discrepancy between the behavior of the program and that of a person. On the one side, the program notices differences in the length of the contour much more often. On the other side, even quite simple problems requiring measurement of the contour length are difficult for a person (people prefer to formulate distinguishing rules in other terms).

In the given problem the length of the contour of figures was broken up into two lots. The pictures of one class ended up in one lot, the pictures of the other class, in the other lot. This distinguishing rule, consisting of three successive operations (contour isolation, measurement of the length, and separation by threshold), is not more difficult than the distinguishing rule, for example, in Problems #6 and #7. Yet for people these two problems present considerably less difficulty.

Problem # 24. Training on the test material.

This is an example of use of the operator *length of lines* in a more complicated problem. To find the solution it is necessary to first get rid of the small-point noise, analogously to the way it was done in Problem #17. Then we must perform convex hull filling on the remaining image, isolate the contour, and measure its length. In correspondence with this distinguishing rule the program attributes the two pictures on the left of the test material to the left class, and the two pictures on the right of the test material, to the right class.

Solving this problem, people usually also classify the test pictures, even though they feel a difficulty in formulating the distinguishing rule — at least they never formulate it in terms of the contour length of the convex hull.

Problem #25. In this problem, as a result of successive break-ups into lots of subsets of certain collections of numbers, the program finds the distinguishing rule, which is a function of three variables.

First the length of lines is broken up into two lots. Then the slope of the subset of short lines is broken up into lots: three short lines are slanted to the left of the vertical, the others, to the right. Then it turns out that the *x*-coordinate of the set of the short lines slanted to the left can be broken up into two lots: two lines are situated on the left part of the raster; the other eight, on the right part. The collection of Boolean numbers corresponding to the second lot is the distinguishing one, "to the right class belong the short lines slanted to the left of the vertical and situated in the right part of the raster."

Problem #26. The massive parts of the figures in the pictures differ by their position. In order to define their position, the program first fills the contour and then applies the measuring operator *coordinates of the center of gravity*. Then the separation by threshold of the *x*-coordinate gives us the classification principle.

Problem #27. The description of the distinguishing rule here includes the coordinates of the centers of gravity of the individual parts of the picture. Therefore the program first separates the pictures by connectedness, measures the coordinates of the centers of gravity and breaks up into lots the *x*-coordinates (see Fig. 9). This operation produces at the output three collections of Boolean numbers of the second level, corresponding to the three lots into which the *x*-coordinates of the centers of gravity were broken up. After this, as a result of applying to the second collection of Boolean numbers the *logical operator*, the program gives on the first level the collection of Boolean numbers corresponding to the statement "in the picture there exists a fragment situated in the middle of the raster (it belongs to the second lot)". This is the distinguishing collection of Boolean numbers.

Problem #28. This problem differs from the previous one in the following: what are broken up into lots here are not the *x*-coordinates of the whole set of fragments, but only those of the subset of "white" (contour) figures. The second distinction, which consists of a considerable variety of the pictures-fragments, was irrelevant from the point of view of the distinguishing rule, but influenced the entire search during the training. As a result the program stores in memory a large number of collections of Boolean numbers (reflecting this variety of pictures) and, consequently, spends much time processing all these collections. The behavior of the program while solving this problem is described in detail in §5.2. Here we should notice one more difference between the behavior of the program and that of people in solving problems. What is meant here is the difference in difficulty of classification principles that use the information about the position of figures on the raster. It has turned out that the subjects immediately notice the mutual position of figures or their parts, but have difficulty finding the distinguishing rules based on the absolute positions of figures. The program defines the mutual positions of figures with the help of the operator *comparison*, whose arguments are the absolute coordinates of the centers of gravity of each figure. However, it starts comparing the pairs of parameters only after having investigated the possibility of building the distinguishing rule on the basis of each of these parameters separately. Thus the program, unlike people, prefers as simpler precisely those distinguishing rules that include absolute coordinates.

Problem #29. This is one more variant of the classification principle that uses absolute positions of figures. In order to single out the subset of the outer points, the program first applies the operator *contour filling*; as a result, the inner points get erased and in every picture there will be only two fragments left. Then, after separating the pictures into

fragments, application of the operator *contour isolation* gives at the output two collections of Boolean numbers defining the subsets of the continuous figures and contour points. All the distinguishing rules built by the program in the process of solving this problem are described in §5.2.

The search tree for this problem is defined by two factors. First, it depends greatly on the material given (on the problem itself). Indeed, the appearance in machine memory of, for instance, a collection of Boolean numbers entails the use of several operators: *logical operator*, operator *union*, etc. Each one of these operators can, in its turn, store in memory new output collections of objects. For example, the use of such an operator as *union* (with all the unavoidable re-drawings and measurements that follow) immediately produces a large number of such collections. And each one of these collections will, in its turn, require the use of some operations, etc. If we compare this search for the distinguishing rules with the search for the way out of a labyrinth, our situation can be represented as a changing (constantly re-built) labyrinth, where each step along a passage can add new passages to this labyrinth. Generally speaking, these additions are different in different problems, and as a result the search goes on in different labyrinths. Therefore, it is the problem itself that defines what the program will do: whether it will wander in the dead ends — re-built as a result of its work — or build only bridges leading to the goal.

On the other hand, at each crossroads of this labyrinth the choice of the path is defined by the order of execution of operators as given to the program. The change of this order may lead to considerable changes of solution time for the same problem. On the other hand, an order chosen once and for all may be convenient for solving certain problems (those in which the variants leading to solution are checked first) and inconvenient for others.

A good example of this situation we find in Problem #29. The pictures contain few elements, and the elements exhibit little variety. As a result, there are few collections of Boolean numbers in memory (see Table 2). Therefore the search should not be long. However, the order of searching turns out to be very inconvenient: among all the possible variants of application of the operator *separation by connectedness*, its application to pictures — the result of work of the operator *contour filling* — happens last; hence we end up with a very long solution time.

Problem #30. An example of the situation unfavorably influenced by the two factors mentioned above: on the one side, the order of searching does not correspond to the problem; on the other side, during the processing of all the hopeless variants the program stores in memory a large number of collections of Boolean numbers which in their turn require processing.

Problem #31. This is a very complicated problem, both for people and the program. In order to solve it, parts of parts of pictures must be analyzed. Indeed, the first separation of pictures by connectedness gives on the second level eleven pictures-fragments, each one depicting one black figure with little white lines. In order to isolate these lines, the operator *contour isolation* must be applied to the pictures-fragments; then they must be again separated by connectedness. The sequence of operations which led to the separation of pictures into classes was as follows: the whole set of fragments obtained as a result of previous operations was broken up into two lots by the length of the contour — there are

short (contours of little white lines) and long (outside contours of figures) parts. After this, the slope of the subset of parts with the short contour is broken up into lots; the little lines may be only vertical or horizontal. The union of the last ones gives at the output on the second level the collection of pictures-fragments, each having only contours of horizontal lines. From the sum total of eleven pictures-fragments, nine will have three horizontal lines and two will be empty — the correspondent fragments have no horizontal lines. After the hull filling and the subsequent measurement of the area, the *threshold operator* notices that the area can be broken up into three lots: a "zero" lot for two empty pictures, as well as a "small" and a "big" lot (defined by whether the small lines are situated compactly or far away from one another). Finally, the *logical operator*, using one of the output of the *threshold operator* — the collection of Boolean numbers of the second level corresponding to the "small" area — forms on the first level a collection of Boolean numbers that correctly separates pictures into classes. These Boolean numbers can be interpreted as the truth-values of the statement given in the caption of the Problem.

Problem #32. The distinguishing rule: divide the picture by connectedness; measure the coordinates of the centers of gravity; separate fragments (into the left ones and the right ones) by applying the *threshold operator* to the *x*-coordinate of the centers of gravity (the program finds the threshold value of the *x*-coordinate via the procedure of breaking up into lots); apply the operator *union* to all the right fragments; apply the measuring operator *area* to the pictures obtained as a result of the previous operation; separate them into big and small by applying the *threshold operator* to the result of the previous operation. To the right class belong the pictures with small area (unions of all the right fragments of the pictures).

The program, while searching for this distinguishing rule, before separating the fragments of pictures into left and right ones, separates them into big and small ones (the area of fragments is broken up into lots). Consequently, the program performs separately the union of big and small fragments (which in itself does not lead to finding a solution, yet takes up much time) before performing the union of the right fragments.

Problem #33. Four pictures were added to the training collection, to facilitate the solution. Because of the differences in the size of individual pictures-fragments in the additional pictures, the area of figures cannot be broken up into lots, and the program solves the problem more quickly.

Problem #34. One of the simplest problems using the operator *comparison*. The distinguishing rule: fill the contour; measure the area of the obtained picture; apply the operator *hull filling* to the pictures; measure the area of the pictures obtained as a result of the previous operation; if the ratio of these areas is small, then the picture belongs to the left class.

Problem #35. The distinguishing rule is similar to the previous one: measure the area of the pictures; fill the convex hull; measure the area of the pictures obtained as a result of the previous operation; if the ratio of these areas is small, then the picture belongs to the right class.

Problem #36. In this case the arguments of the operator *comparison* are the area of the convex hull of the sum total of small points and the area of the convex hull of the sum total of lines. If the ratio of there areas is small, the picture belongs to the left class.

The structure of complex drawing operators (Fig. 14) necessary to isolate points and lines is described in §3.8.

Problem #37. In this problem the operator *comparison* compares the slopes of the figures, one of which is made up by vertical spots and the other, by horizontal spots. If the slopes of these figures are equal then the picture belongs to the right class.

In order to isolate each one of such figures the program first divides the pictures into fragments and measures the slope of the obtained figures. The slope breaks up into two lots. As a result the *threshold operator* produces (on the second level) two collections of Boolean numbers defining the subset of horizontal and vertical spots. Applying the operator *union* to the first one of these collections we get on the first level pictures with only horizontal spots. Then a convex hull of the obtained images is filled and the slope is measured. An analogous sequence of operations applied to the second collection of Boolean numbers (of the second level) allows us to define the slope of the figures composed of vertical spots.

Problem #38 Unlike in the four previous problems, which also used the operator *comparison*, in this problem the arguments of the operator *comparison* are collections of numbers of different levels. One argument (a collection of numbers of the first level) is the slope of the figures created from the input images by the operation *convex hull filling*; the second argument (a collection of numbers of the second level) is the slope of figures obtained after dividing the pictures into fragments. The application of the operator *comparison* to these collections of numbers gives at the output two collections of Boolean numbers of the second level defining the subsets of the fragment-segments parallel and perpendicular to the big axis of a whole. After this, the application of the *logical operator* to the first of these two collections of Boolean numbers allows us to divide the pictures into classes: “to the left class belong the pictures in which there exists a segment oriented along the axis of the figure”.

Problem #39. To build the distinguishing rule the program must divide the pictures into fragments by boundaries. The language of the program’s elementary operators does not allow us to build distinguishing rules of the following type: the given fragment of the figure is *connected* to this or that fragment. However, the comparison of the mutual position of fragments (using the operator *comparison*) allows the program to describe the distinguishing rule in this problem in terms such as “close–far”. The program gets the arguments necessary for the operator *comparison* in the following way: 1) application of the operator *contour filling* (on the first level) and the subsequent measurement of the coordinates of the centers of gravity give the vector of coordinates (two collections of numbers) of the center of gravity of the figures; 2) analogously, the division of pictures into fragments and the subsequent measurement of coordinates give the vector of coordinates of the centers of gravity of the fragments; 3) contour filling (after the division of pictures into fragments) and the subsequent contour isolation give a collection of Boolean numbers defining the subset of individual segments (analogously to Problem #2). The Boolean numbers of this collection correspond to the truth-values of the statement “the given part stays contour even after contour filling”. Applying the operator *comparison* to these three collections we see that the distance between the centers of gravity of the segments and of the entire picture breaks up into two lots. As a result the operator *comparison* gives at the output two collections of Boolean numbers (of the

second level) defining the subsets of the segments situated either close to the center of gravity of the figure, or far from it. The application of *the logical operator* to each one of these collections of Boolean numbers produces on the first level two collections of Boolean numbers each one of which turns out to be distinguishing. The first one of them gives the following classification principle: “if in a picture there exists a segment situated close to the center of gravity of the figure, then the picture belongs to the right class”.

Problem #40. The same collection of pictures as Problem #39. The classification principle is very simple and does not use division into fragments; the left class differs from the right one in that it contains contour pictures.

Problem #41. This problem is similar to Problem #39. One of the possible solutions is: the segments are situated to the left or to the right of the figures. In fact, even before separating the pictures by boundaries, the program finds another solution of this problem. It uses the comparison of the coordinates of the centers of gravity. The program divides the pictures into fragments and then measures the coordinates of the centers of gravity of the obtained figures twice — after contour filling and after convex hull filling. It can be noted that the directions into which the centers of gravity are shifted as a result of hull filling will be different for different figures. Consequently, the application of the operator *comparison* to this pair of vectors of the coordinates separates the figures into two subsets, and the subsequent application of the *logical operator* gives the following classification principle: “to the left class belong the pictures containing at least one of the figures whose center of gravity gets shifted to the left after convex hull filling”.

Problem #42. In this problem the massive black figures represent “noise”. The program gets rid of it by dividing the pictures into fragments and then applying operator *union* to the contour fragments (small points). The application of the operator *convex hull filling* to this union of points produces horizontal arrows pointing to the right in the pictures of the left class and to the left in the pictures of the right class.

The proposed language of elementary operators lacks such human expressions as “arrows pointing to a certain direction”, so the program formulates the distinguishing rule in terms, accessible to it, though not completely human. The program measures the coordinates of the centers of gravity of the figures obtained as a result of the operations described above. Then it applies to these coordinates the operator *contour isolation* and again measures the coordinates of the centers of gravity of the obtained contour figures. It is the application of the operator *comparison* to the pair of vectors of coordinates that leads to the separation of the pictures of the training collection into classes.

Problem #43. The branch of the search that leads to the distinguishing rule in this problem looks like this: the program consecutively divides the pictures into fragments by connectedness and by branching nodes. The measurement of the length of thus obtained lines (parts of the pictures) and the subsequent application of the *threshold operator* produce at the output three collections of Boolean numbers (of the third level) defining the subsets of individual points, short segments and long segments. The breaking up into lots of the slope of the subset of the long segments separates segments into those tilted to the left and those tilted to the right (of the vertical). The *logical operator* (applied to the objects of the third level and supplying the objects to the first level) gives us the

classification principle corresponding to the statement “in the picture of the left class there exists a long segment tilted to the left”.

Problem #44. Training using the test material. The solution of this problem is described in §5.4.

Problem #45. The straight lines crossing the figures make the solution of this problem more difficult. In order to erase them the program first builds a complex drawing operator, whose structure is described on page 38 in §3.8, *Complex operators*. Application of this operator to the pictures of the training collection produces at the output pictures that differ from the input ones by the lack of the bothersome straight lines (Fig. 16). Then the application of operator *contour filling* separates the closed lines from the non-closed ones and thus separates the pictures of the left class from the pictures of the right class.

Problem #46. Just as in the previous problem, the straight segments here represent noise, and after dividing the pictures into fragments (by connectedness) the program analogously isolates and unites the subset of the oblique lines. Then the convex hull filling, measurement of its thickness, and the break-up into lots by this parameter isolate the subset of pictures in which the oblique line segments build a thin elongated figure. The corresponding collection of Boolean numbers correctly divides the pictures of the training collection into classes.

Problem #47. The final stage of solution is the same as in the previous problem; the program must isolate the pictures in which the small figures build a straight line. In order to isolate the subset of the small figures the program first divides the pictures by connectedness, then isolates the contour, then divides by connectedness again. The length of the contour of the fragments obtained on the third level breaks up into two lots: the figures with the long contour and the figures with the short contour. After this, in order to build the distinguishing rule the program uses the subset of pictures with the short contour.

The longer solution time for this problem is explained by the fact that the program first goes along a wrong path. After the first division of pictures into fragments the area of these new pictures is broken up into two lots. The slope of the subset of “big” figures — a parameter, irrelevant for the distinguishing rule — in its turn is broken up into tow lots (figures with the vertical orientation and figures with the horizontal orientation) and thus strongly influences the further search. Then the program tries to apply the operator *union* (with all the possible subsequent re-drawings and measurements) separately to the vertical big figures and to the horizontal big figures, and spends a long time doing this.

Problem #48. Four pictures were added to the previous problems in order to exclude the unnecessary breaking up into lots of the slope of the big figures (on the second level). As a result the program finds the same distinguishing rule in a shorter time.

6. CONCLUSION

A system capable of learning on a small number of examples must necessarily have a complex enough initial organization. It has to possess a large quantity of *a priori* data about its environment. In the program described here the *a priori* information is coded in the language, and not only in the sum total of the operators, but also in the rules of search. Experiments with the program have shown that the chosen language is, indeed, to a certain degree adequate for the world of "human" visual problems.

In §5.5 we pointed out some discrepancy between the behavior of the program and that of people (even in the problems the program had been created to solve). Some of these discrepancies can be explained by the poverty of the list of operators. Indeed, sometimes the program, while solving some problems, very complicated for people, cannot learn to distinguish a triangle from a square. The addition to the program of new operators must enhance its capabilities.[6] One more difference in the behavior of the program and that of people is due to the fact that the program thinks "too deeply" — the depth of search for the distinguishing rules is too great. Such discrepancies are easy to get rid of. However, the further refining of the program in this direction will add little to our understanding of the process of visual perception. In our opinion, today the problems of modeling are centered not in the sphere of plain black and white geometric problems (such as M. Bongard's ones [5]).

The latest developments in computers resulted in publication of works that expose "theories of thinking" in the form of computer programs [11]. These programs deal mostly with solving logical and mathematical problems and proving theorems, that is, they work in the areas where conscious psychological processes play an especially important role. Therefore it becomes possible to widely use self-observation and special psychological tests [12] to get information about the algorithms used by people. Such a method of theorizing in the domain of visual perception is complicated by the fact that for people the processes connected to solving visual problems are mostly unconscious. A good example of this is the automatic "lighting correction" in the mechanisms of constant color perception [13], which occurs totally unconsciously (and hence remained outside of the scope of attention of researchers for a long time).

Up to the present moment there have been several attempts at modeling the process of visual recognition training. There were large discrepancies between the behavior of the program and that of people in case of such programs as *Perceptron* [14] or *Geometry* [15]. It was obvious that these programs learn very differently from how a person does, and therefore cannot serve as a basis for building a theory of learning (training). On the other hand, the behavior of the program described here (while solving a limited class of problems) was close to human. The main shortcoming of the program (as a model of the process of recognition training) consists in that it can solve problems from too narrow an area (as compared to people's), even though probably still considerably larger than the

[6] In particular, we have to give to the program a possibility to divide lines into separate points, measuring in each point the slope and the curvature of the line. Then these parameters can be broken up into lots according to the general scheme. In case of a successful break-up, the subsequent union of the points of one of the lots would allow us to get individual segments of lines with this or that characteristic: the application of the operator *union* to the points with a constant slope will allow us to isolate individual sides of polygons; the application of the operator *union* to the points at which the line has big (and, separately, small) curvature will allow us to distinguish between the vertices and the sides of polygons.

world of *Perceptron*'s problems. This can also be a cause of the discrepancy in the behavior of the program and that of people even within the domain of these problems. Indeed, having limited the created program to learning only the recognition of flat images, we cannot be always sure that in psychological experiments people will be solving precisely this problem. Therefore what we need now is not any further refining of the program, but rather some corrections on the level of formulating the problem, deepening our understanding of the problems which people put and solve on the level of visual perception.

Bibliography

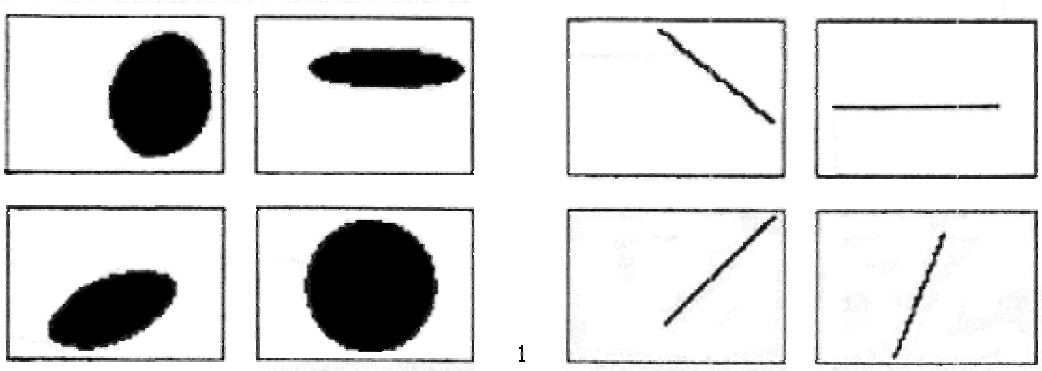
1. *Structural Methods of Recognition and Automatic Reading*. Ed. A.I. Mikhailov, VINITI, Moscow, 1970.
2. Muchnik I.B. *Algorithms of formation of local characteristics for visual images.* LiT, Vol. 10, 1966.
3. Evans T.E. *A program for the solution of a class of geometrical analogy intelligence-test questions*, in *Semantic Information Processing*, MIT Press, 1968, pp. 271–353.
4. Minsky M.L. *Steps toward artificial intelligence*. Proc. IRE, Special computer issue, 1961, pp. 8–30.
5. Bongard M.M. *The Problem of Pattern Recognition*. Moscow: Nauka, 1967.
6. Maksimov V.V., Bongard M.M. *A program learning the classification of geometric objects*, in *Pattern Recognition. Adaptive Systems*. Moscow: Nauka, 1971
7. Maksimov V.V. *Modeling of the process of pattern recognition*, in *Processing of Visual Information And Regulation of Motor Activity*. Sofia, 1971.
8. Maksimov V.V. *A program learning the classification of geometric objects. Language and experiments.* in *Structural Methods of Recognition and Automatic Reading*. Ed. A.I. Mikhailov, VINITI, Moscow, 1970.
9. Zavalishin I. V., Muchnik, I. B. *Linguistic (structural) approach to the problem of pattern recognition*, A&T, 1969, No. 8.
10. Automated Analysis of Complex Patterns. Moscow: Mir, 1969.
11. A. Newel, H. A. Simon. *Programs as theories of higher mental processes*. Stray F. W., Waxman B. (eds.). *Computer in Biomedical Research*, v. 11. Academic Press, New York, 1965.
12. Weinzweig, M. N., Poliakova M. P. *A possible approach to the problem of creating artificial intelligence.* In the present book.
13. Nikolaev I. P. *Some algorithms of surface color recognition*. In the present book.
14. F. Rosenblatt. Principles of Neurodynamics, Cornell Aeronaut. Lab. Rep. 1196–G–8; Spartan, Washington, 1962.

APPENDIX

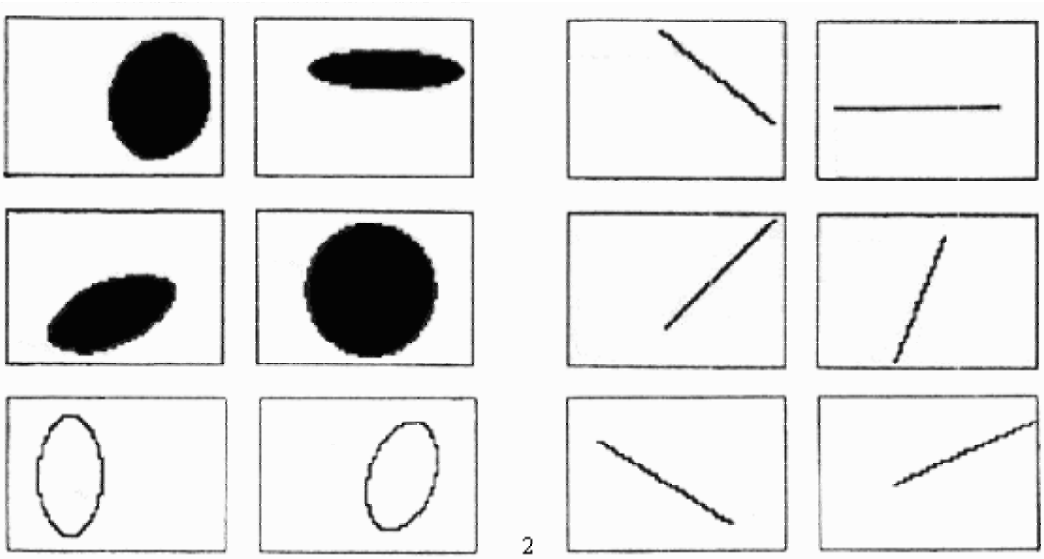
Each one of the 48 problems offered to the system for training contains several pictures broken into two classes — left and right. In some problems, together with the training pictures, the system was offered some test pictures. The test material is given in the bottom part of the corresponding drawings. Below are the classification principles ("answers") usually found for these problems by people.

Problem #1. In the pictures of the left class there are ellipses; in the pictures of the right class, straight line segments.
Problem #2. Training on the test material. Same as Problem #1.
Problem #3. In the pictures of the right class there exists a straight line segment.
Problem #4. In the pictures of the right class there exists a straight line segment outside of the contour.
Problem #5. The classes differ by the slope of the longitudinal axis of the figure.
Problem #6. In the pictures of the right class there are thin ellipses; in the pictures of the left class, "fat" ellipses.
Problem #7. In the pictures of the left class there are long ellipses; in the pictures of the right class, short ellipses.
Problem #8. In the pictures of the left class the white figures are oriented vertically; in the pictures of the right class, horizontally.
Problem #9. Training on the test material. Same as Problem #8.
Problem #10. Training on the test material. In the pictures of the left class the image occupies a small area; in the pictures of the right class, a big area.
Problem #11. Training on the test material. The pictures of the right class contain big figures; the pictures of the left class contain small figures.
Problem #12. Training on the test material. The pictures of the left class contain three figures; the pictures of the right class contain five figures.
Problem #13. The pictures of the left class consist of three fragments; the pictures of the right consist of two fragments.
Problem #14. The pictures of the left class contain three black rectangles; the pictures of the right class contain one black rectangle.
Problem #15. In the pictures of the left class there exists one figure containing two black fragments.
Problem #16. In the pictures of the left class there exists one black figure with a hole.
Problem #17. In the pictures of the left class the contour figures are concave; in the pictures of the right class they are convex.
Problem #18. The classes differ by the slope of the longitudinal axes of the areas filled with small points.
Problem #19. In the pictures of the left class there are no points outside of contour figures.
Problem #20. In the pictures of the left class there exists a small figure; in the pictures of the right class all the figures are large.
Problem #21. Same as Problem #20.
Problem #22. In the pictures of the left class there are short lines; in the pictures of the right class, long lines.
Problem #23. See §5.5.

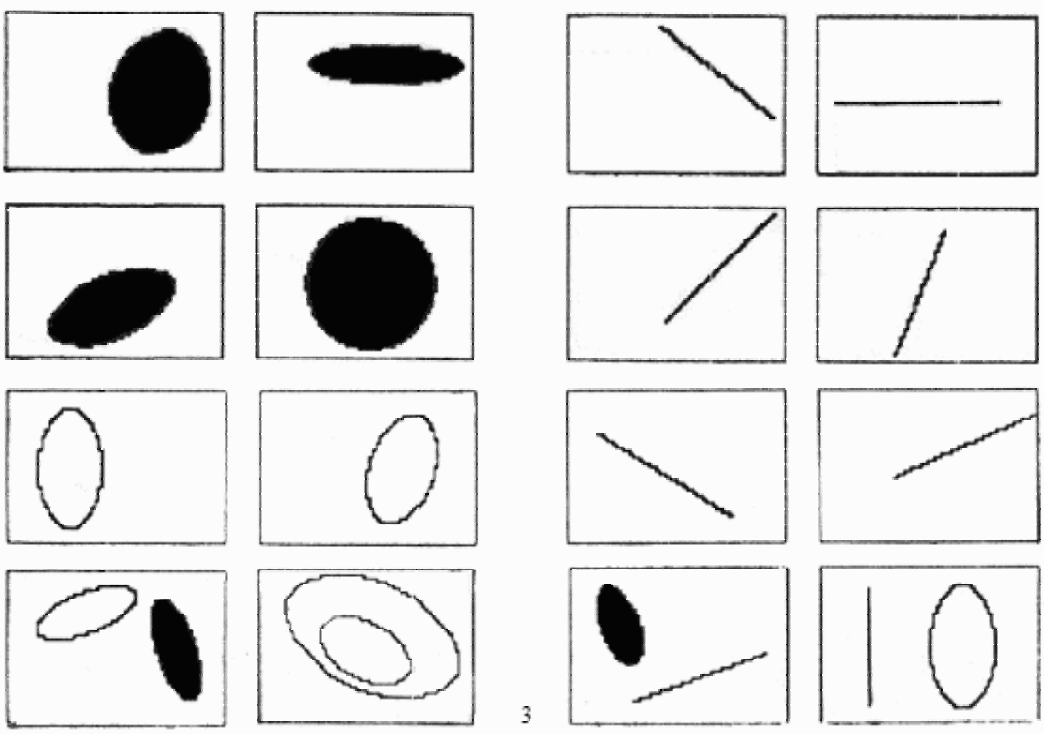
Problem #24. Training on the test material. See §5.5.
Problem #25. In the pictures of the right class there are short segments tilted to the left of the vertical in the right part of the raster.
Problem #26. In the pictures of the left class the figures are situated to the left; in the pictures of the right class they are to the right.
Problem #27. In the pictures of the left class all figures are located around the center of the raster. In the pictures of the right class the figures are located near the edges.
Problem #28. In the pictures of the left class the white figures are situated in the left part of the raster; in the pictures of the right class, in the right part.
Problem #29. The isolated (outside) point in the pictures of the left class is situated in the left part of the raster; in the pictures of the right class, in the right part of the raster.
Problem #30. In the pictures of the left class there is at least one small square (white or black).
Problem #31. In each picture of the left class there exists at least one black figure in which the horizontal white lines are situated in a compact group.
Problem #32. The total area of the figures situated in the right part of the raster is big in the pictures of the left class and small in the pictures of the right class.
Problem #33. Same as Problem #32.
Problem #34. In the pictures of the left class the figures are more concave than in the pictures of the right class.
Problem #35. In the pictures of the left class the figures are "darker" than in the pictures of the right class.
Problem #36. In the pictures of the left class the small points occupy smaller area than lines; in the pictures of the right class, they occupy a larger area.
Problem #37. In the pictures of the left class the two figures consisting, respectively, of vertical and horizontal spots are perpendicular; in the right class they are parallel.
Problem #38. In the pictures of the left class the segments making up the elongated figure are situated alongside the figure; in the right class, perpendicular to it.
Problem #39. In the pictures of the left class the straight-line segments are touching the smaller fragment of the figure ("horns"); in the pictures of the right class, the bigger fragment ("legs").
Problem #40. In the pictures of the left class the figures are white.
Problem #41. In the pictures of the left class the straight line segments are to the left of the figures; in the pictures of the right class, they are to the right of the figures.
Problem #42. The small points form an arrow that points to the left in the pictures of the left class, and to the right in the pictures of the right class.
Problem #43. The big straight-line segment in the pictures of the left and right classes has different slope.
Problem #44. Training on the test material. Same as Problem #43.
Problem #45. Pictures of the left class are open after subtracting the straight lines; pictures of the right class are closed.
Problem #46. In the pictures of the left class the curved lines form a straight line.
Problem #47. In the pictures of the right class the small figures (black and white) are situated on one straight line.
Problem #48. Same as Problem #47.



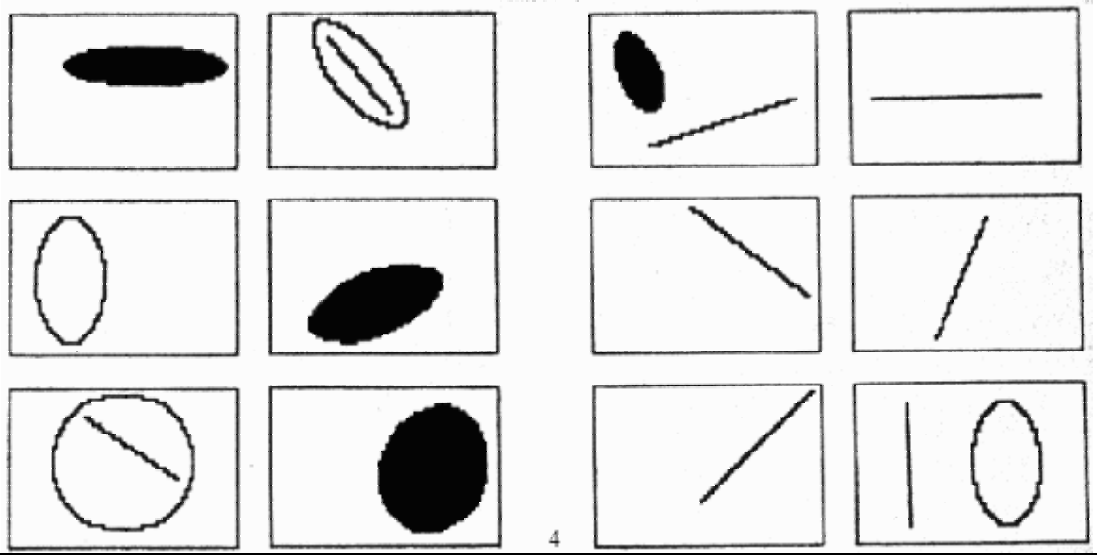
1



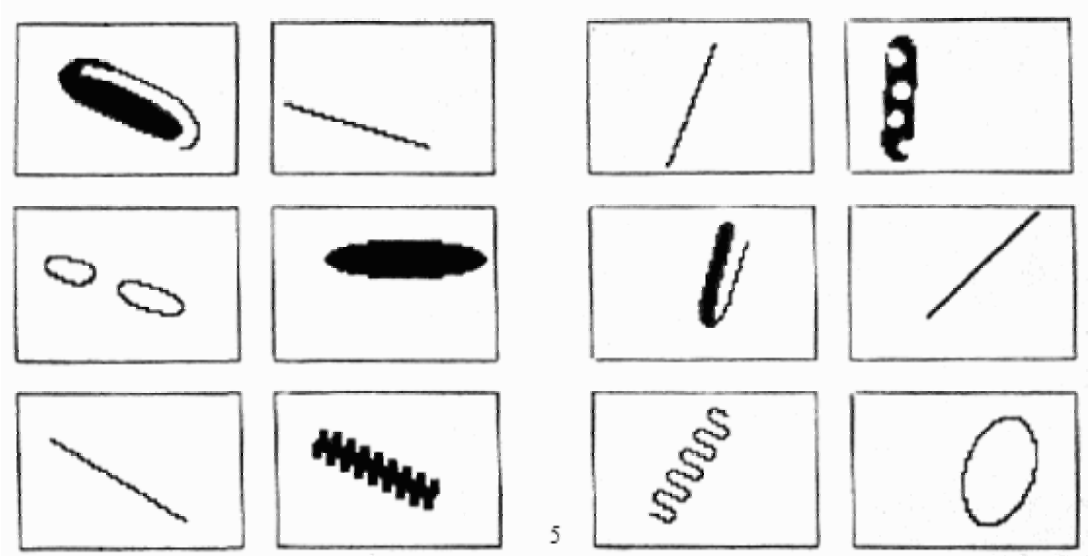
2



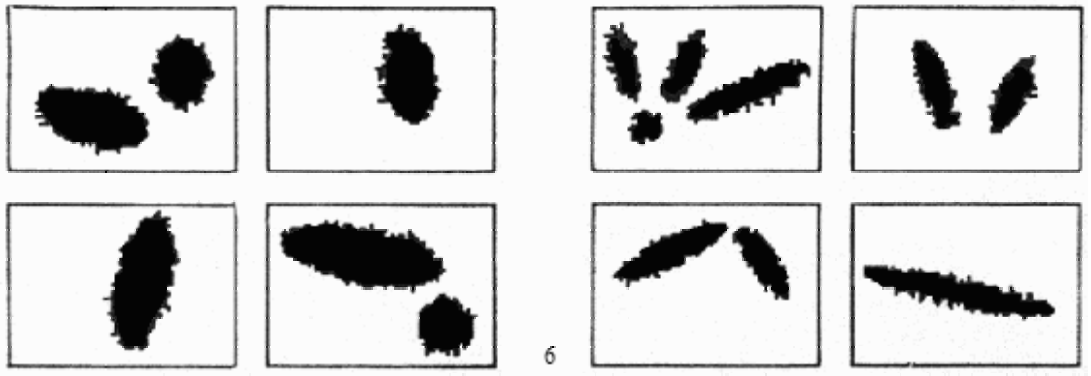
3



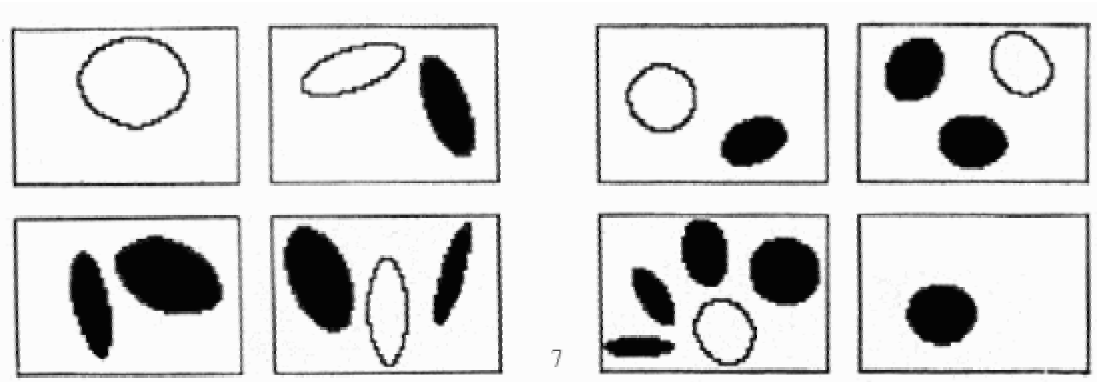
4



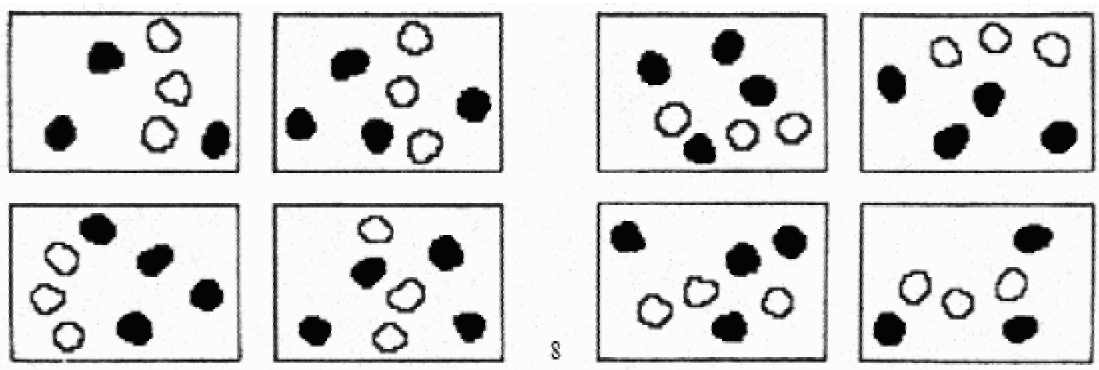
5



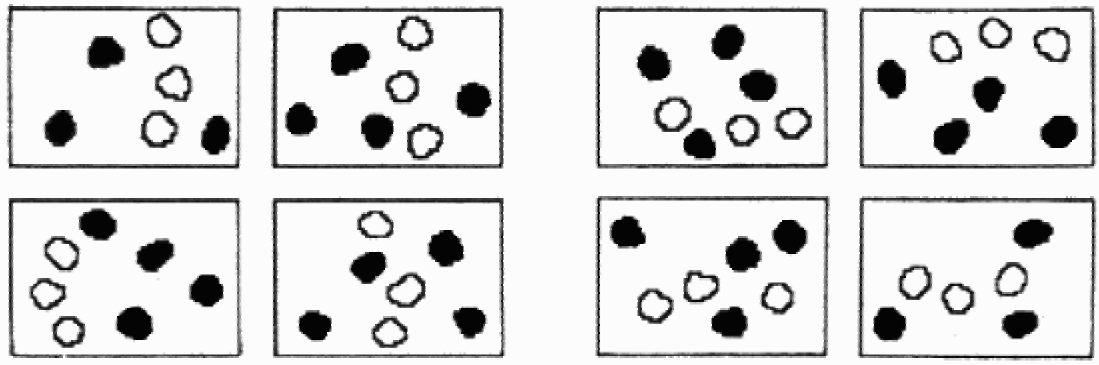
6

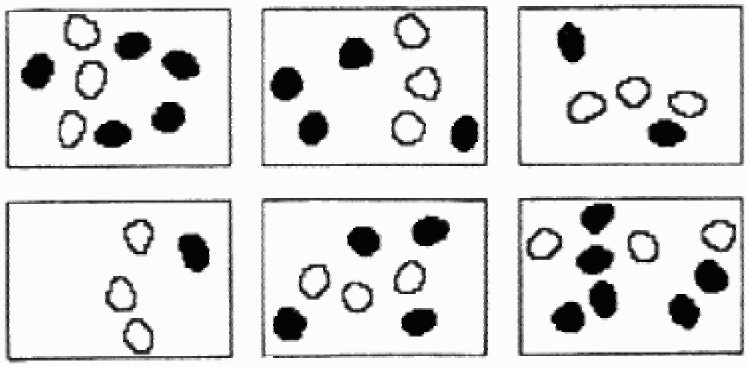


7

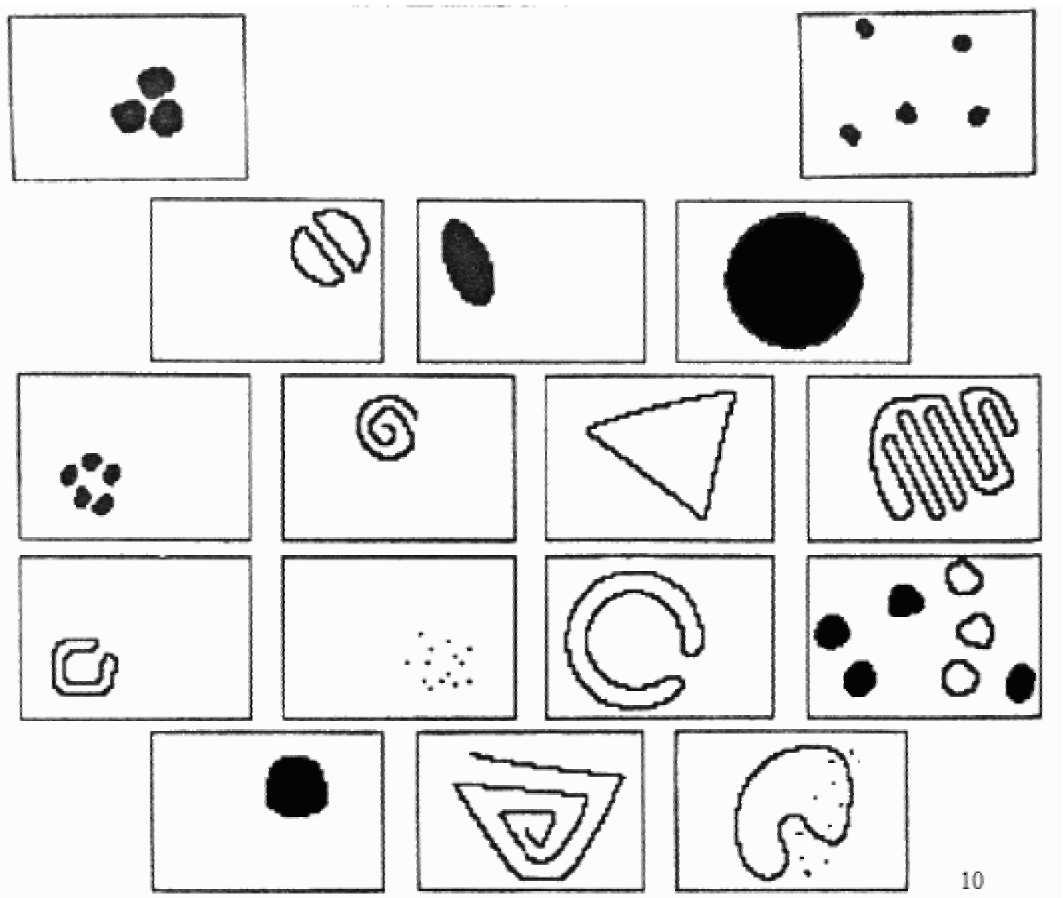


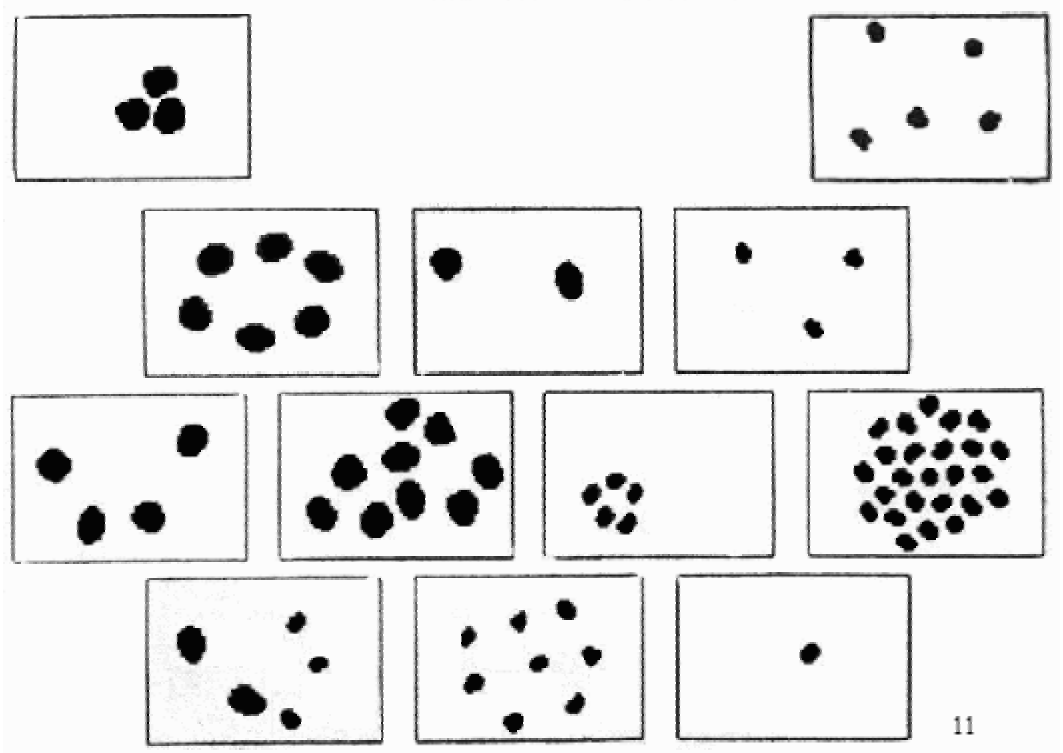
8

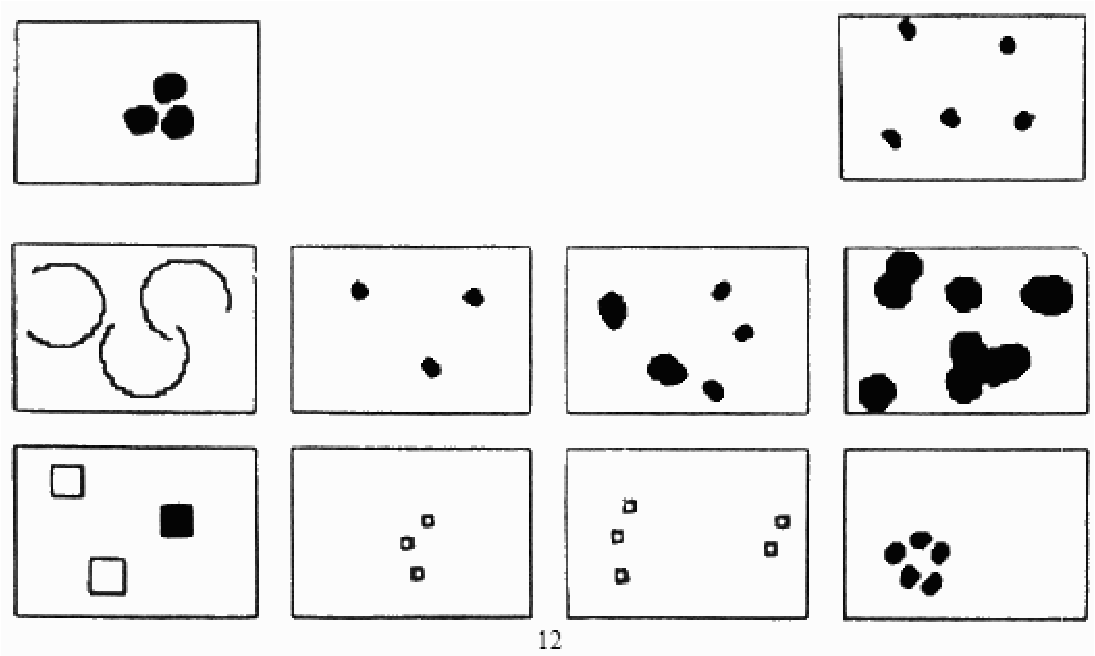




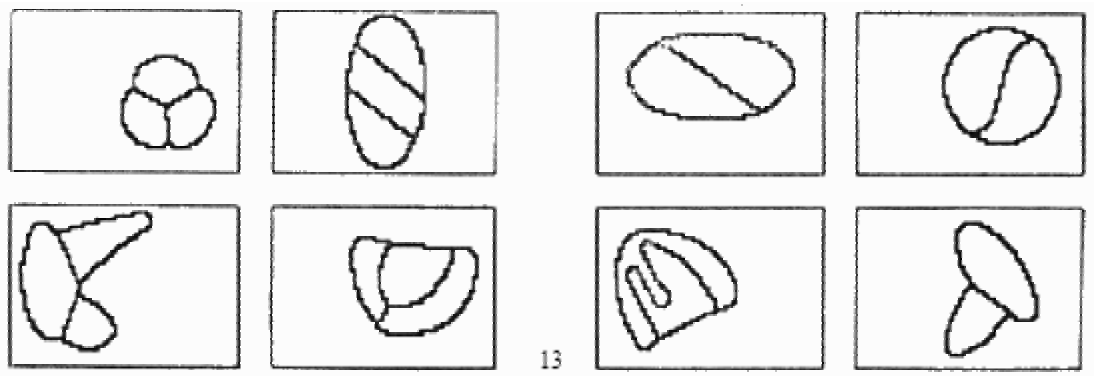
9


10

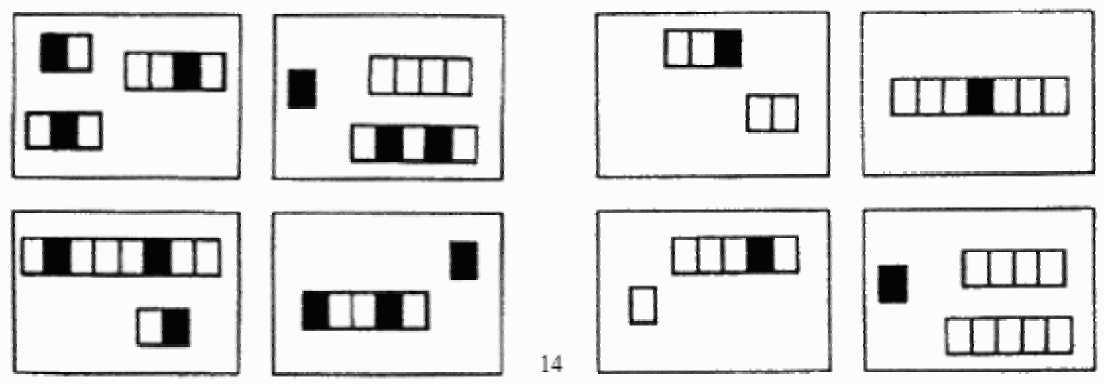

11



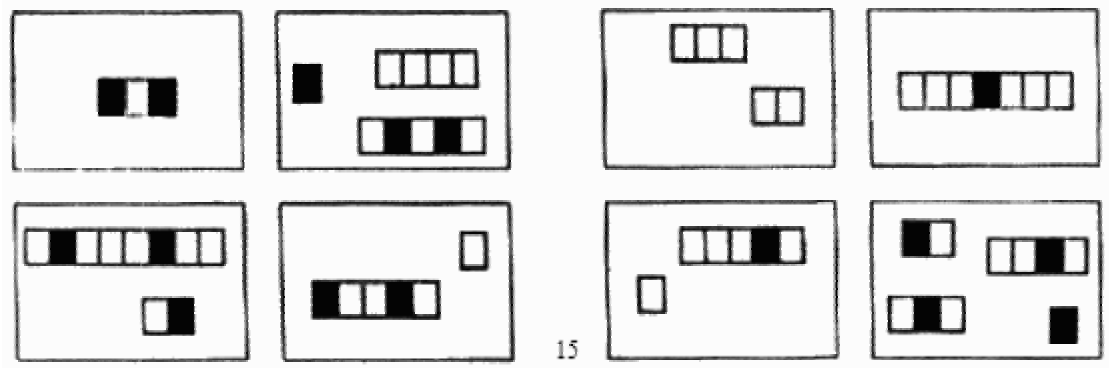
12

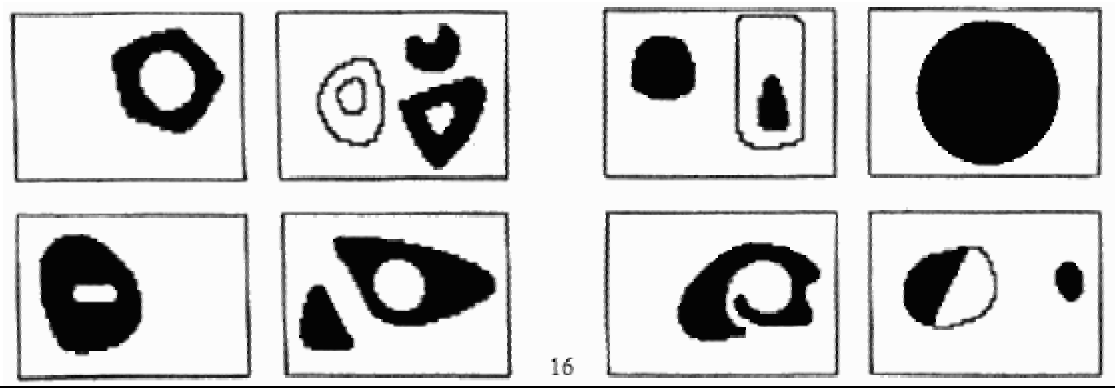


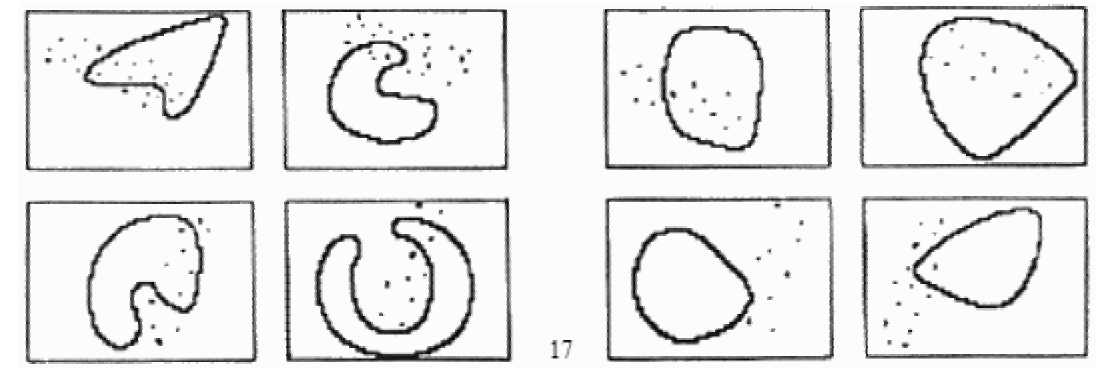
13

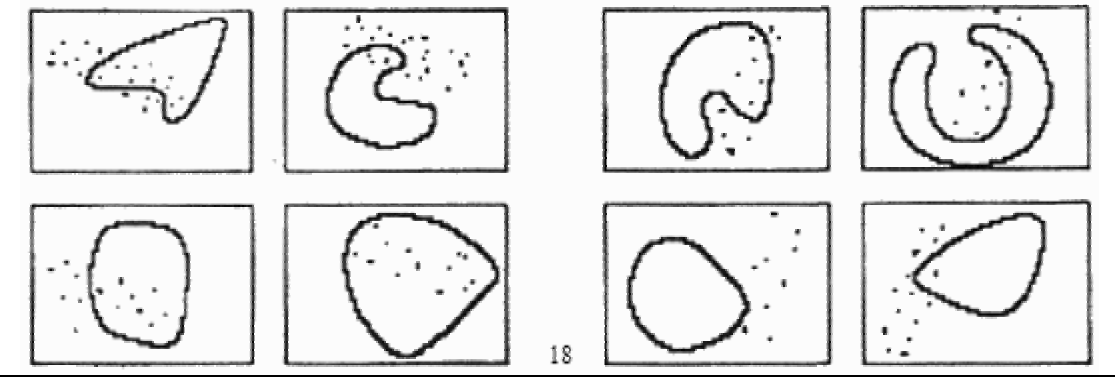


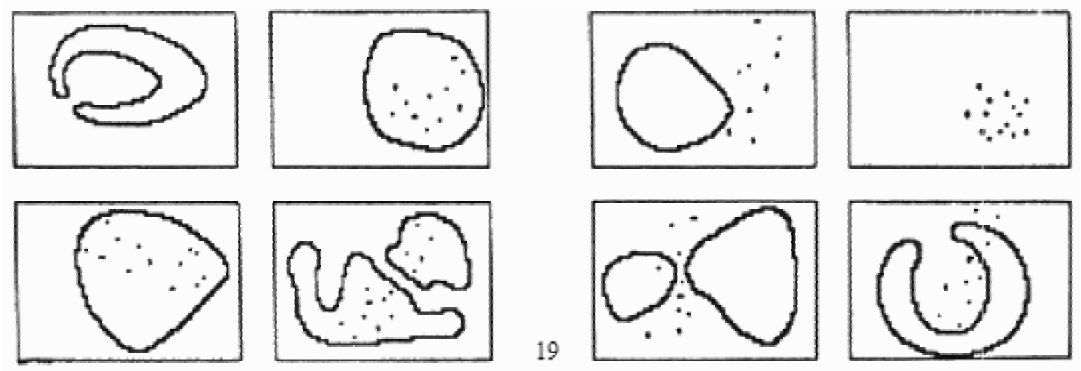
14


15

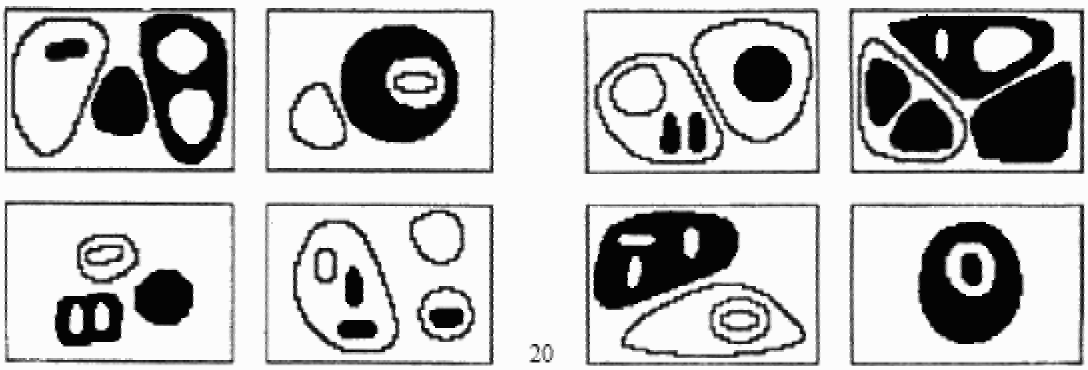

16


17

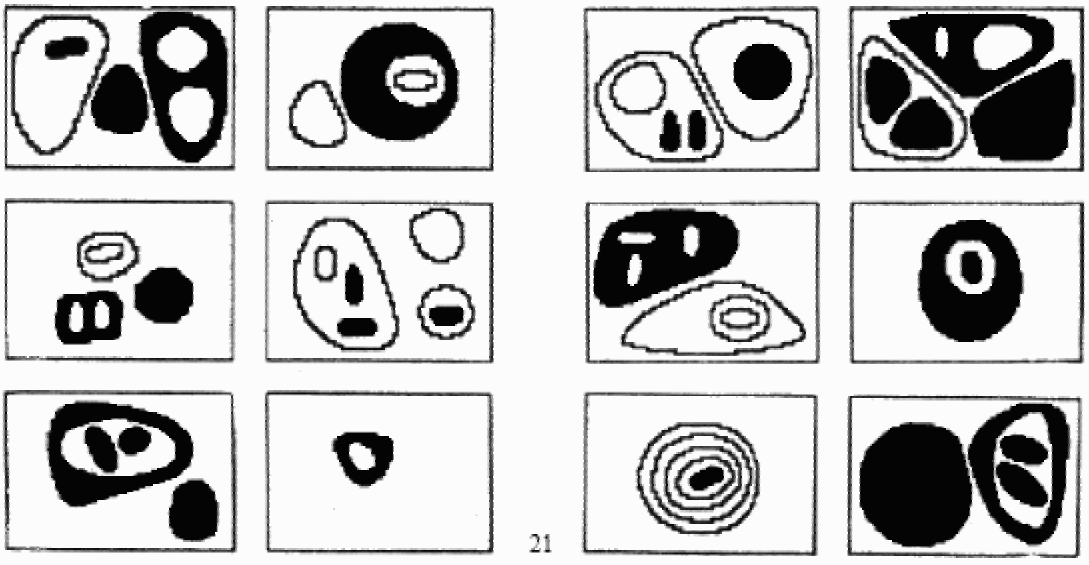

18



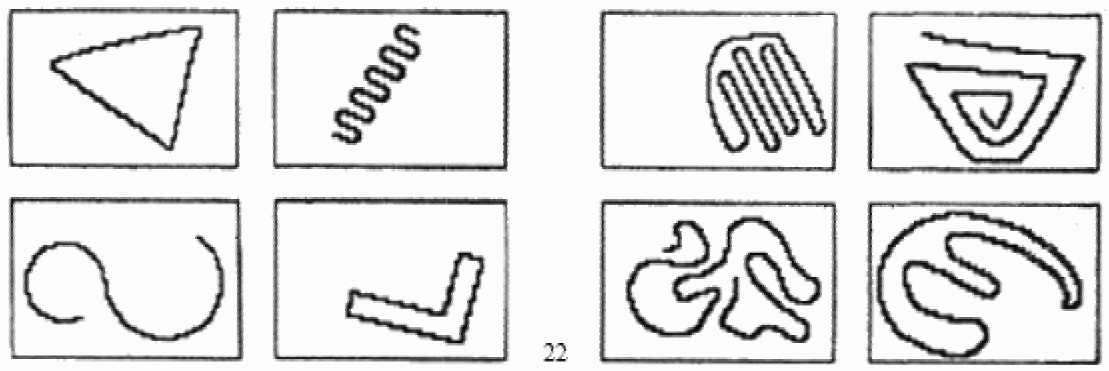
19



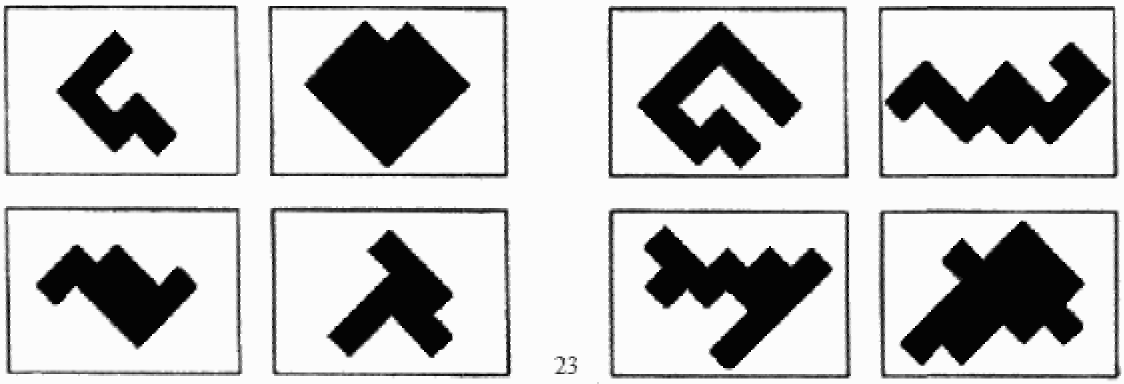
20



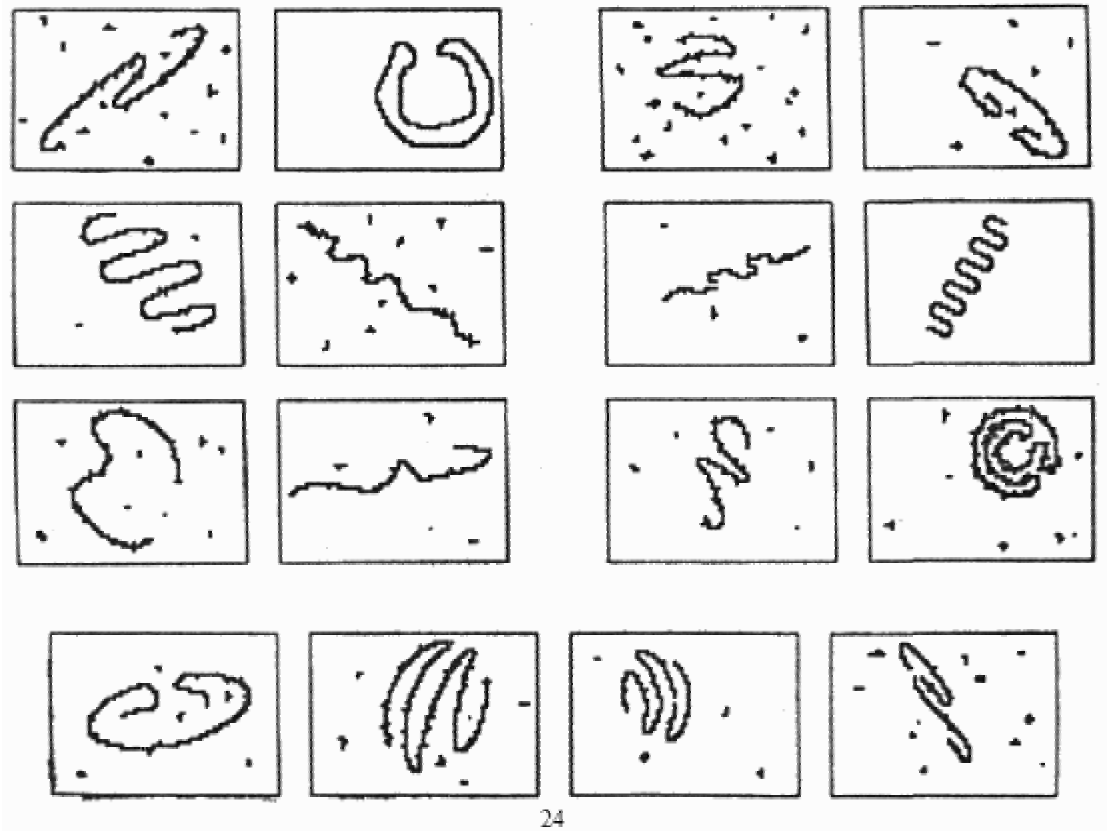
21



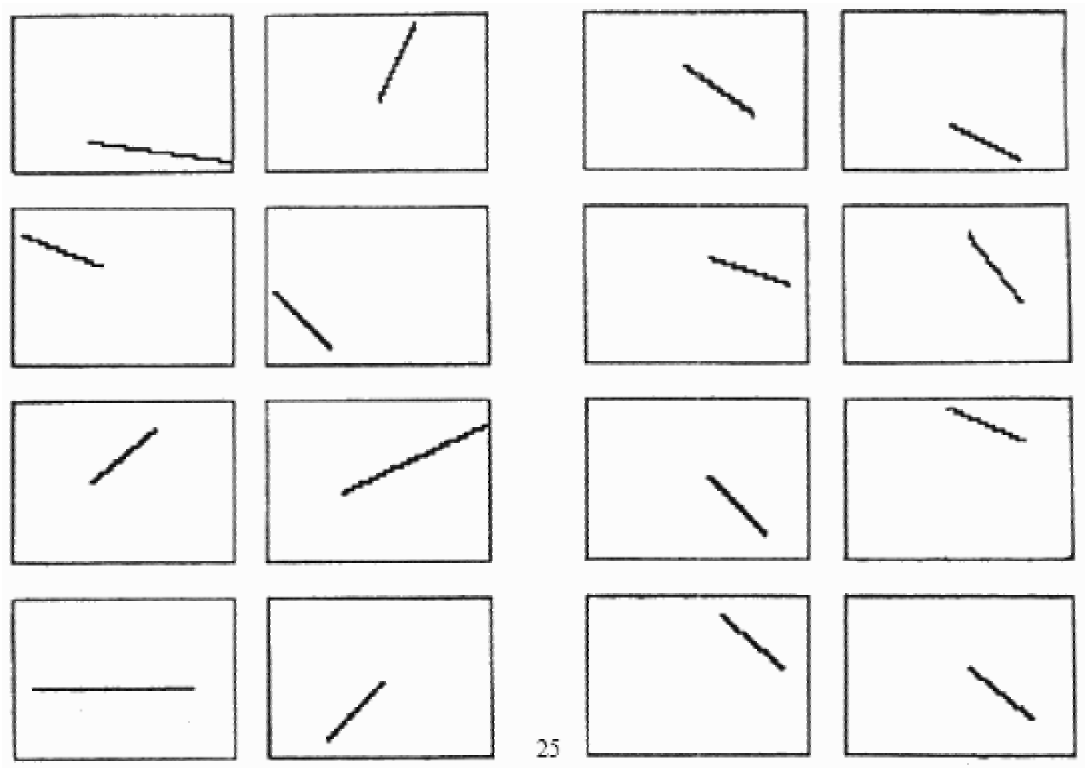
22



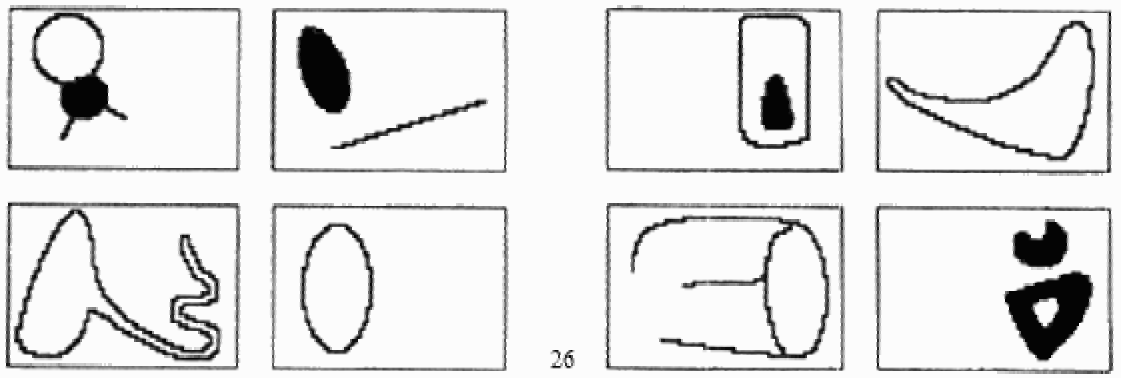
23



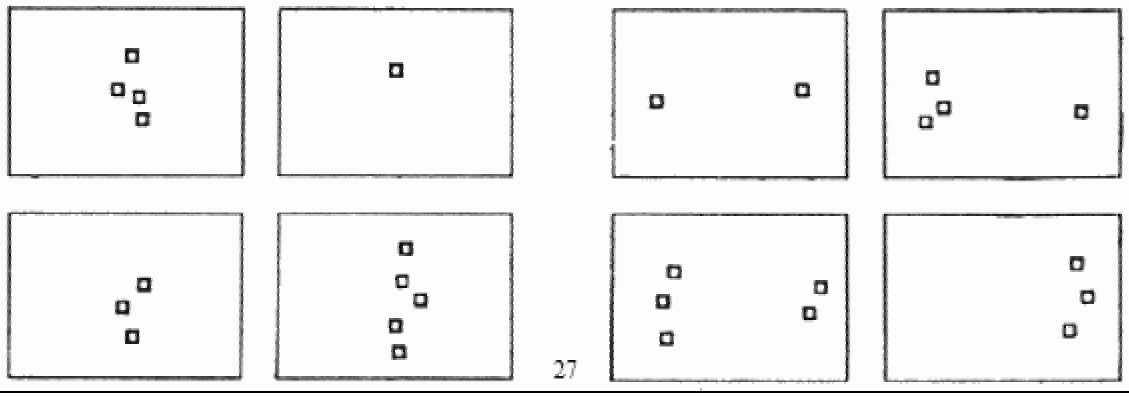
24



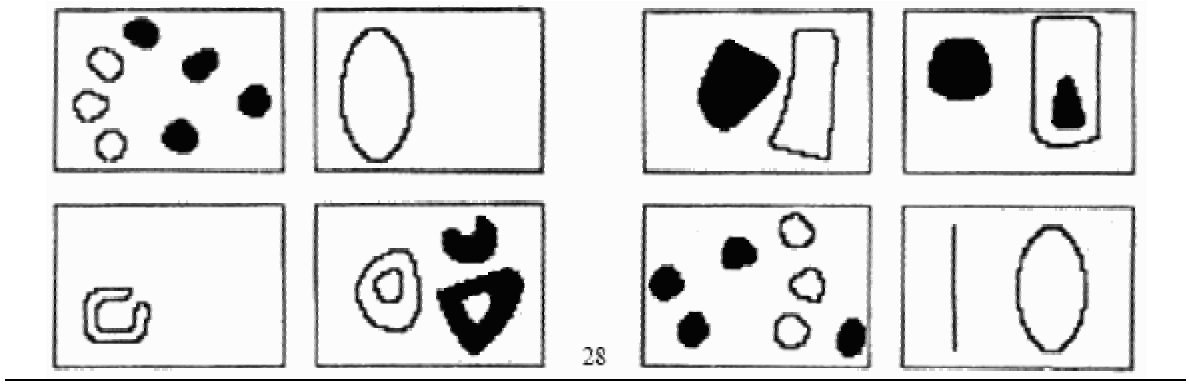
25

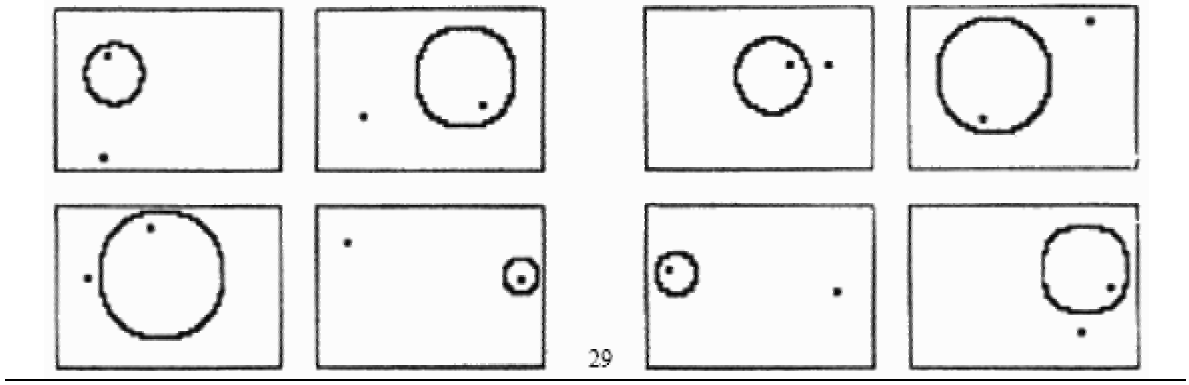


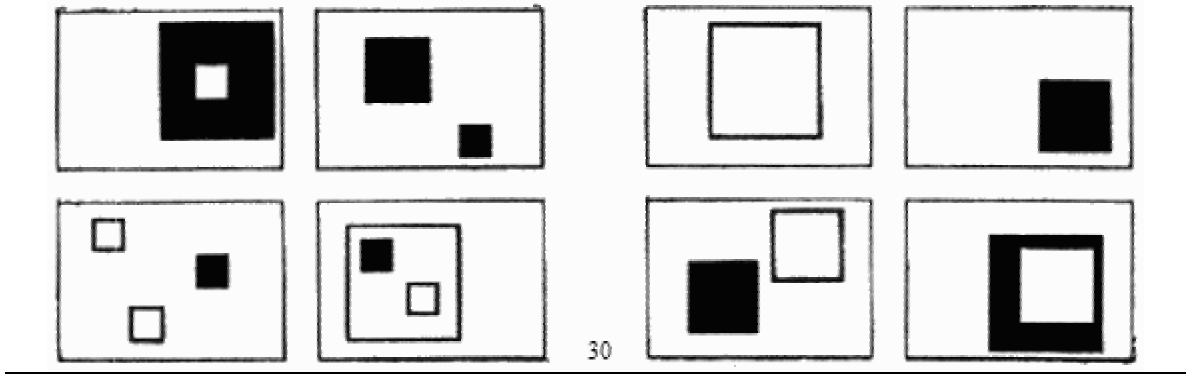
26

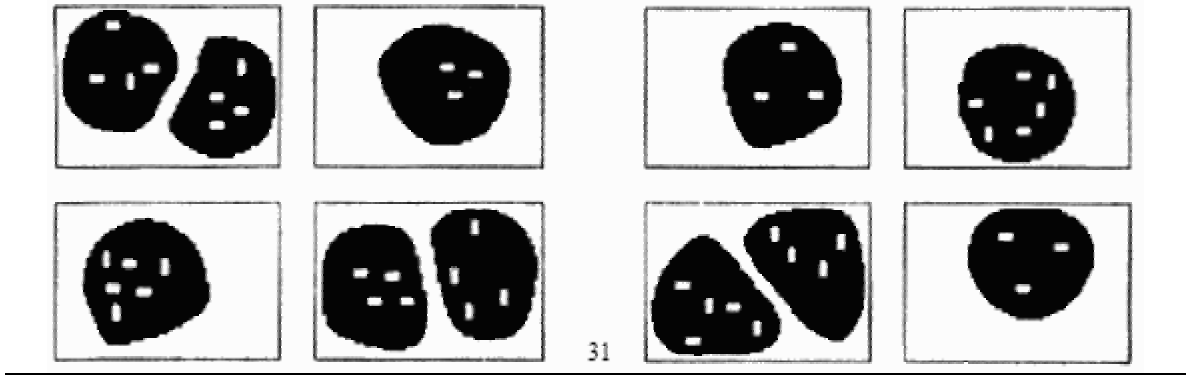


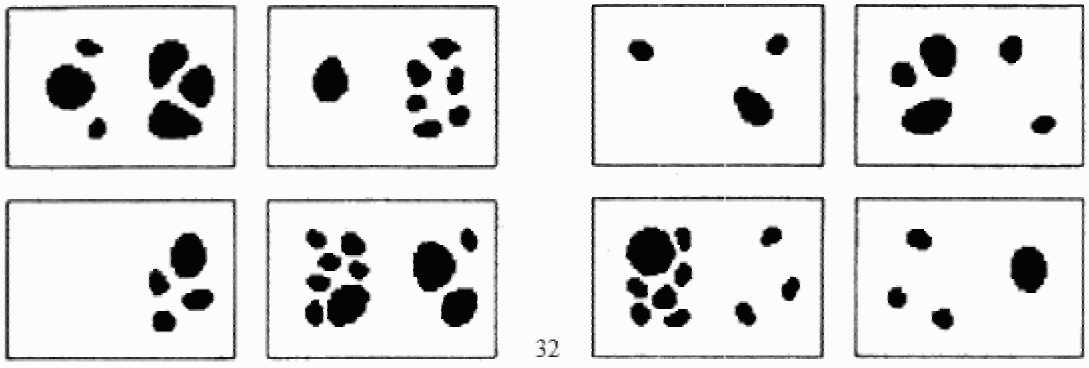
27


28

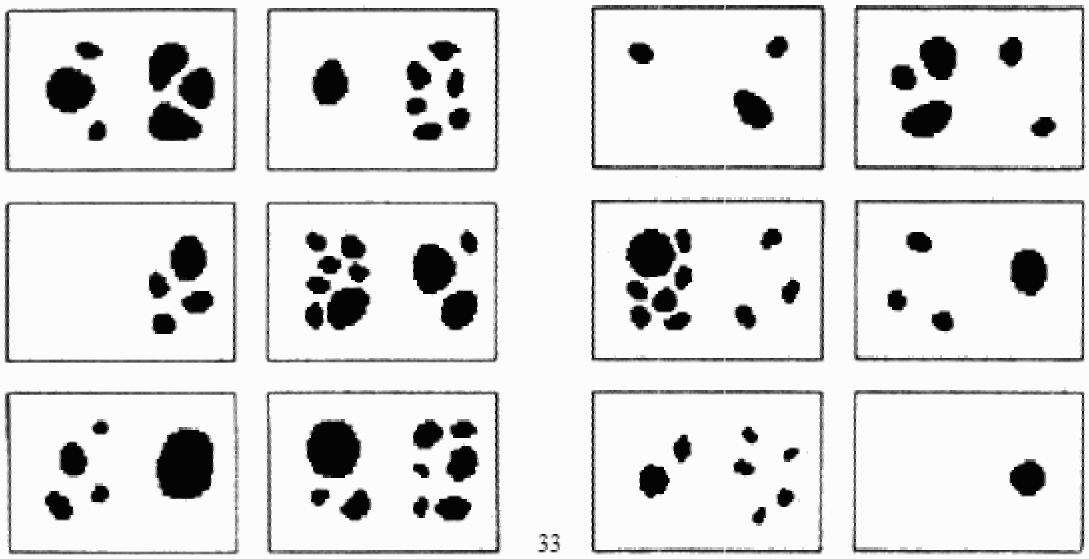

29


30

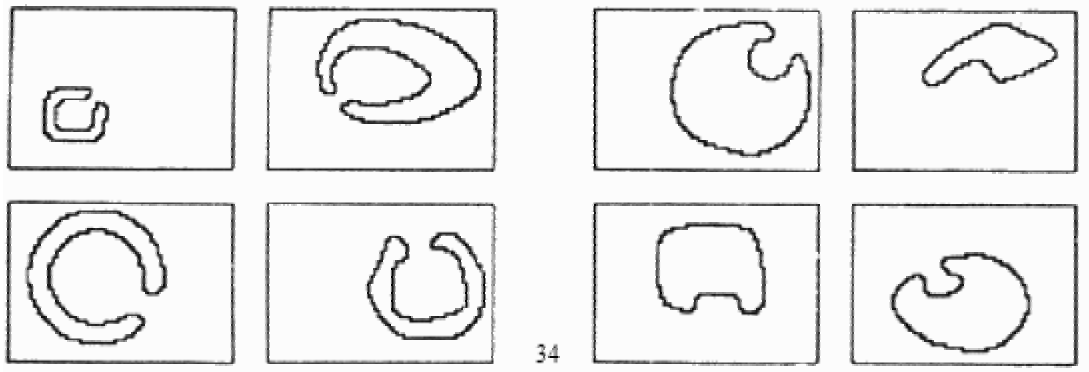

31



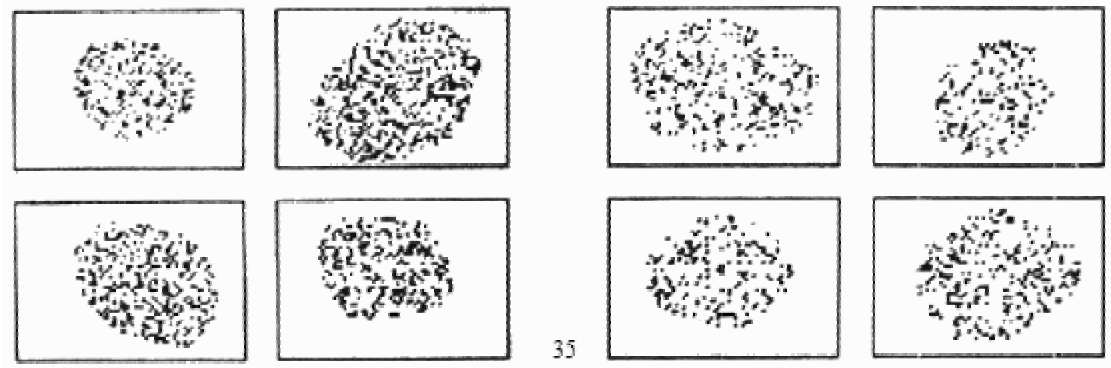
32



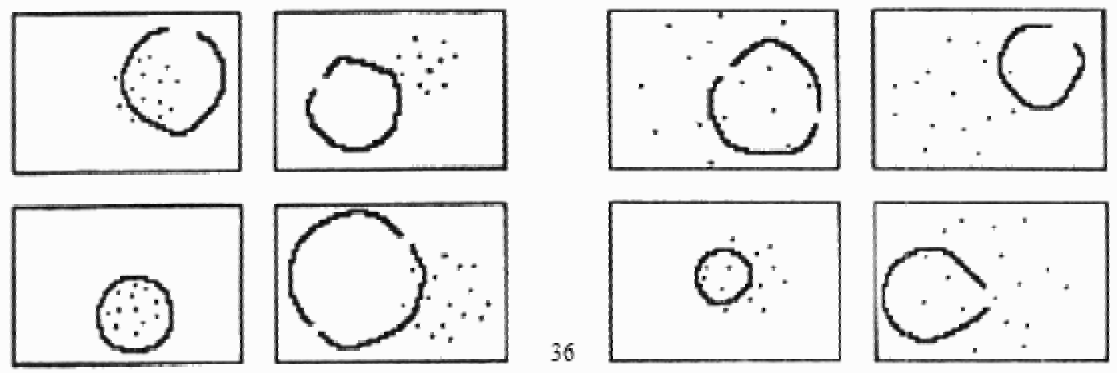
33



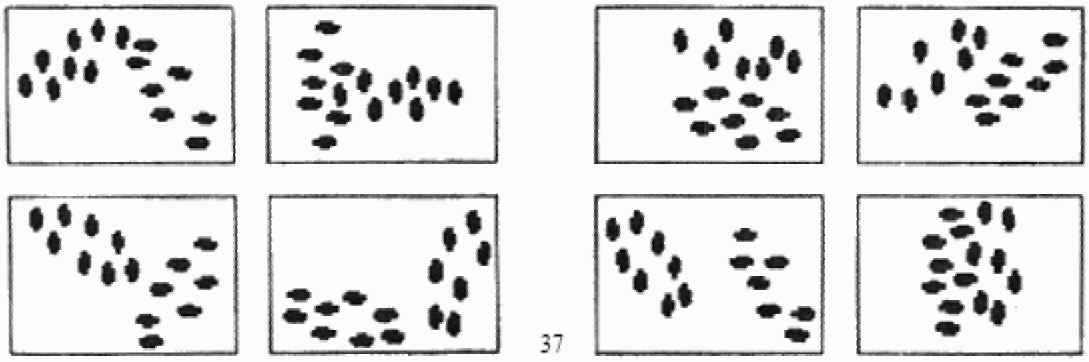
34



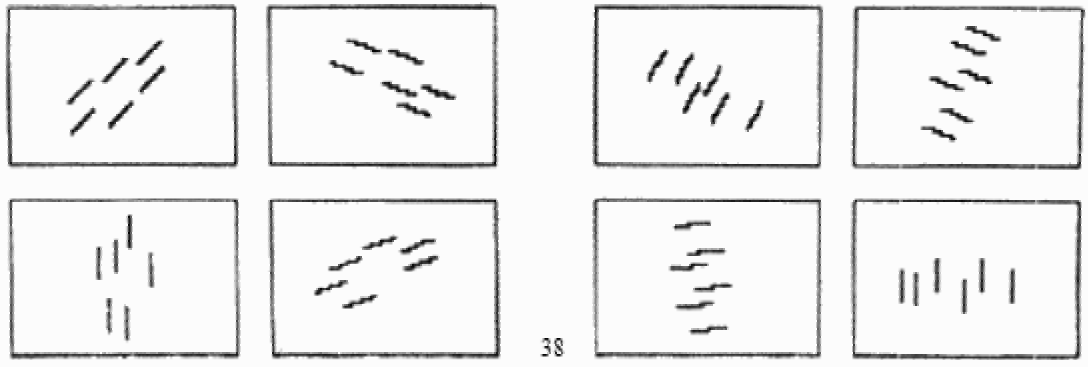
35



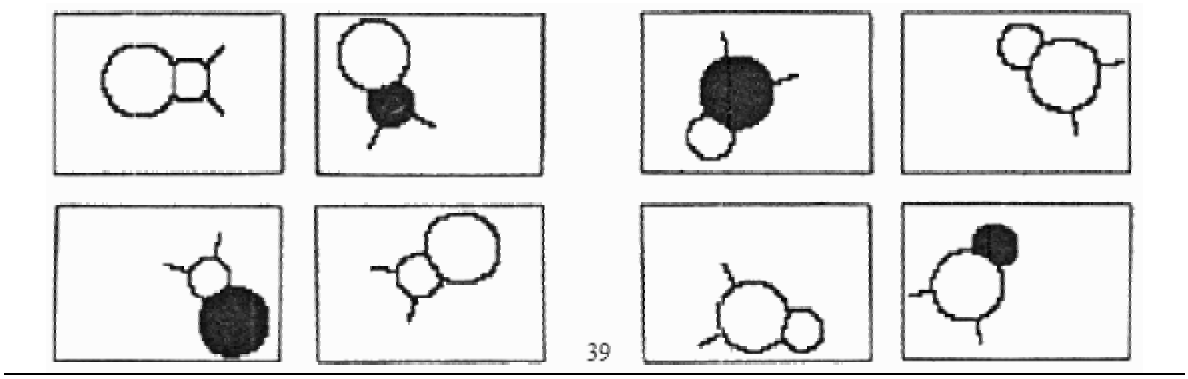
36

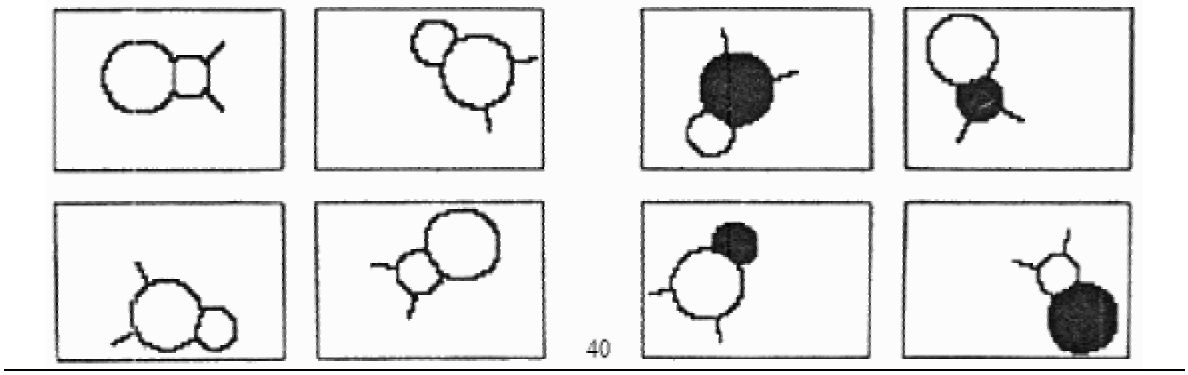


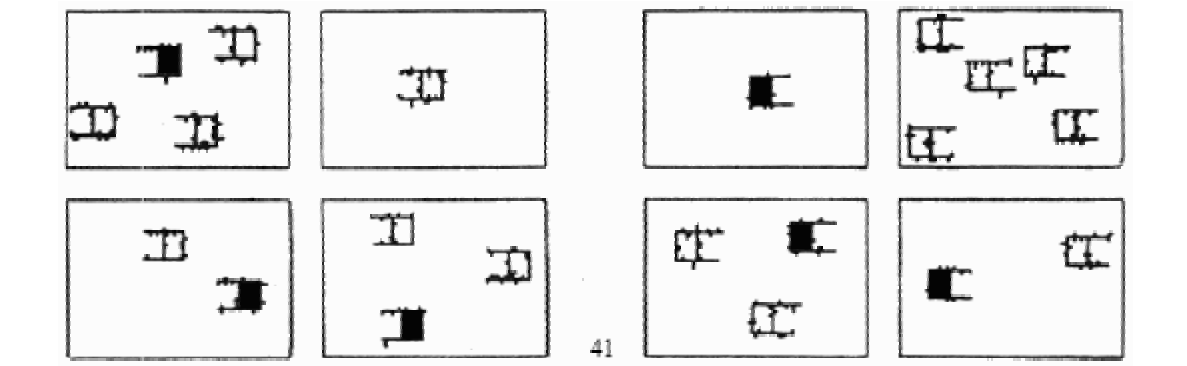
37

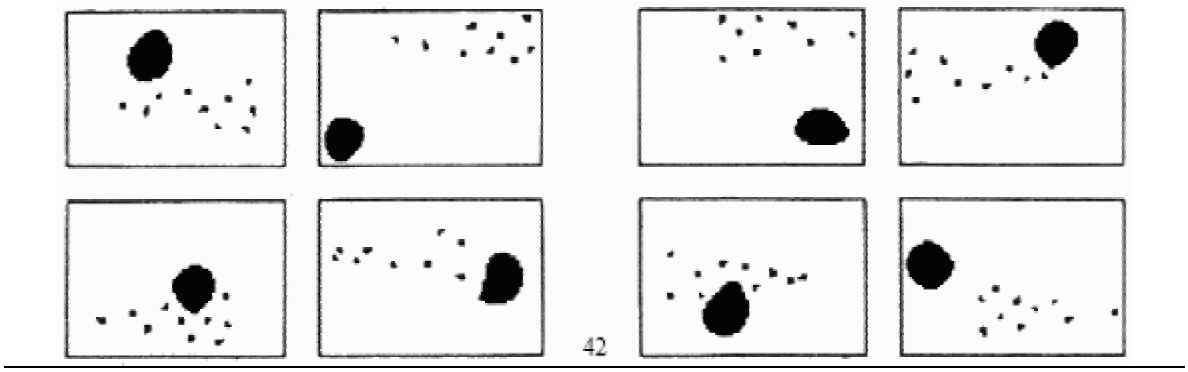


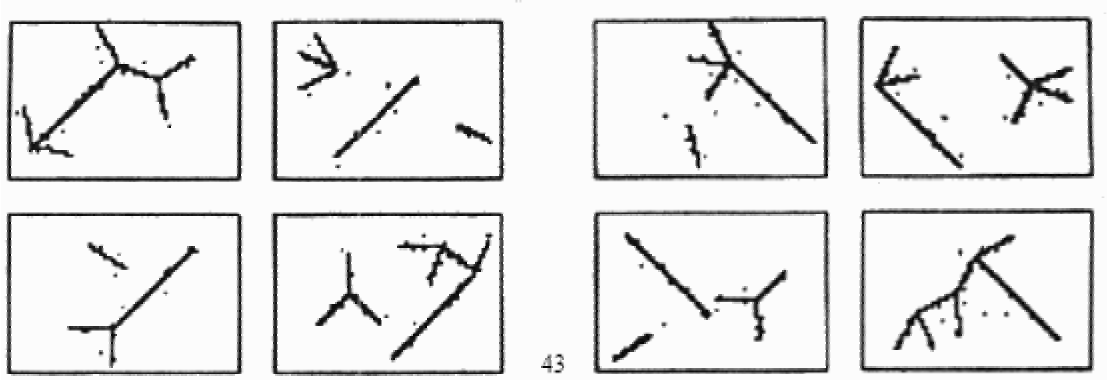
38


39

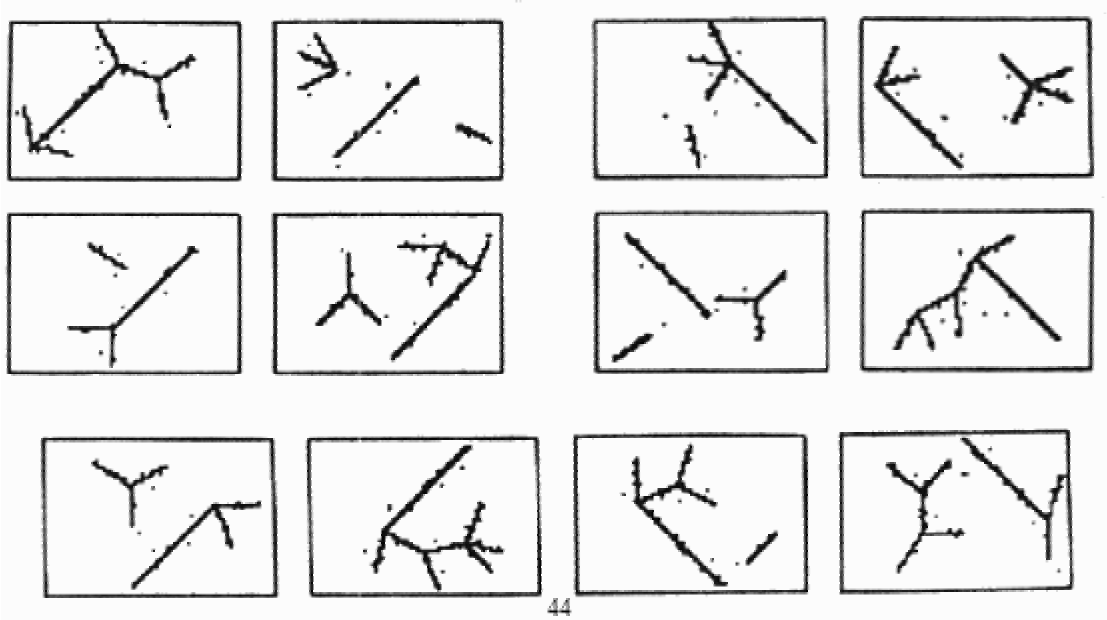

40


41

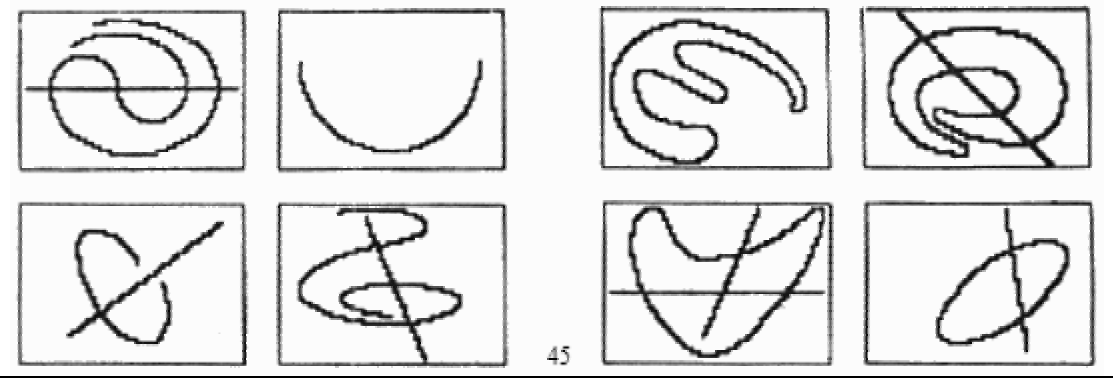

42



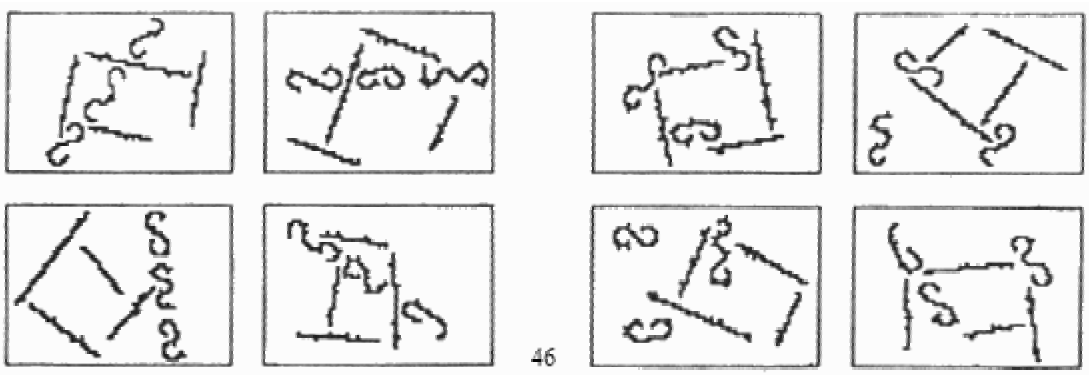
43



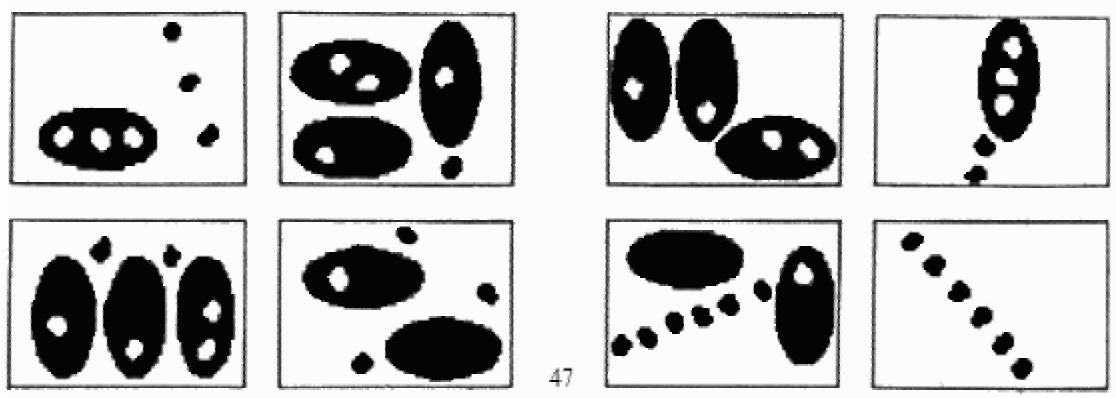
44



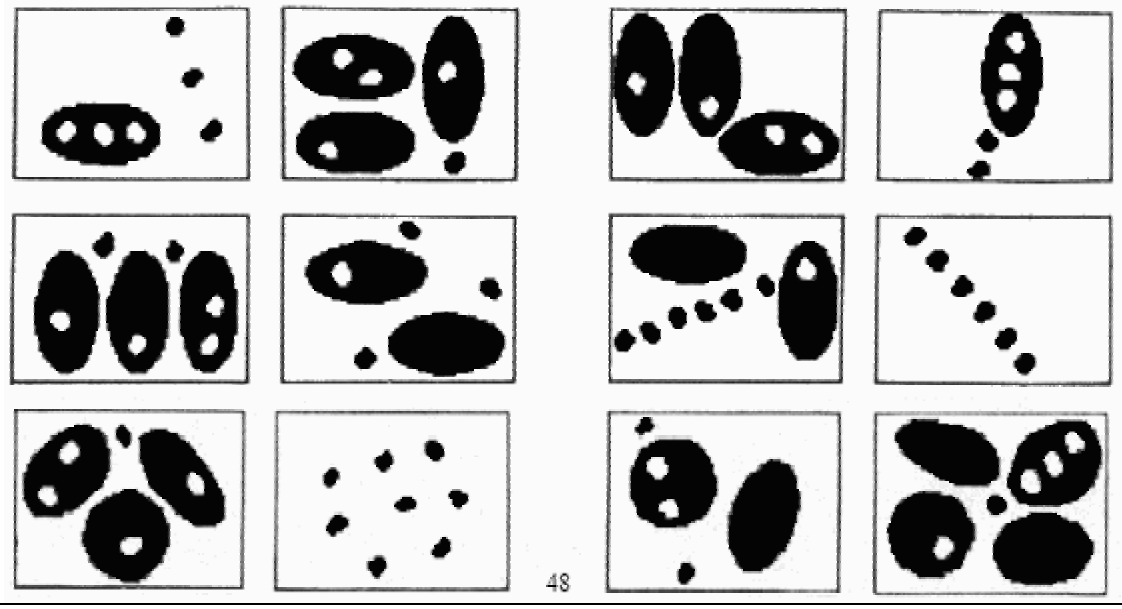
45



46



47



48