STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

CHARACTER RECOGNITION USING MORPHOLOGICAL TRANSFORMATIONS

VASILE PREJMEREAN AND SIMONA MOTOGNA

Abstract. Starting from a digital image representing a character (or, more general, an object), we will obtain formal descriptions of the object structures, formed from primitives and the relations between them, using morphological transformations (thinning, prunning, determining the corners, determining the primitives). From such a description we will construct a description grammar for the object under study, grammars that are used in the syntactical recognition of similar objects. The object being recognized has a corresponding description through morphological transformations that allows us to study if the description belongs to the language generated by the designed grammar.

1. Introduction

Generally speaking, we can say that an image transformation is dedicated to human eyes or is made for a pattern recognition reason.

This paper presents a method for character recognition using morphological transformations, whose purpose is to obtain certain structures formed from lines and curves (type 3 images [6]) needed for pattern recognition. The notion of morphology comes from animal and plants form study, but for us *morphological processing* ([3, 5]), means the determination of the object structures from their images.

2. Morphological Processing

Morphological processings consist in operations through which an object X is modified by a structuring element B, yielding to a form convenient for future processing (pattern recognition). The two interacting elements (X and B) are represented as sets in the Euclidian bidimensional space.

Most of the morphological operations can be defined by two basic operations, erosion and dilation described in the following.

2000 *Mathematics Subject Classification.* 68Q45, 68T10.

1998 *CR Categories and Descriptors.* F.4.2 [**Theory of Computation**]: Mathematical Logic and Formal Languages – *Grammars and Other Rewriting Systems*; I.5.2 [**Computing Methodologies**]: Pattern Recognition – *Design Methodology*.

Notation 2.1.

The translation of B in x denoted by B_x, is the translation of the structural element B such that the origin O_B is located in x.

Definition 2.1.

The erosion of X by B, denoted by $X \ominus B$, is the set of all points x such that B_x is included in X:

$$X \ominus B = \{x | B_x \subset X\}.$$

Remark: Erosion is an operation that decreases the object.

Definition 1.2.

The dilation of X by B, denoted by $X \oplus B$, is the set of all points x such that B_x and X have a nonempty intersection:

$$X \oplus B = \{x | B_x \cap X \neq \emptyset\}.$$

Remark: Dilation is an operation that increases the object.

The two presented basic operations have the following properties: translation invariant, distributivity, local knowledge, iteration, increasing, duality, and so on [3].

Next, we shall present some usual transformations obtained from the two basic operations described above (X^C denotes the complement of X).

a. *Hit-Miss*, denoted by $X * B$, verifies if a structure $B \in X$ and $B^C \in X^C$:

$$X * B = (X \ominus B) \cap (X^C \ominus B^C) = (X \ominus B) \cap (X \oplus B^C)^C =$$
$$= \quad (X \ominus B_{Ob}) \backslash (X \oplus B_{Bk}) \qquad (\textit{we denote } B \textit{ by } B_{Ob}, \textit{ and } B^C \textit{ by } B_{Bk})$$

because from:$(X^C \oplus B) = (X \ominus B)^C$ (duality property for all X and all B) it results that

$$(X \oplus B^C) = (X^C \ominus B^C)^C \quad (\textit{applied to } X^C \textit{ and } B^C). \tag{1}$$

B_{Ob} must be *matched* with the object X, and B_{Bk} with the ***B**ac**k**ground*;

b. *Open* of X relative to B, denoted by X_B is the domain scanned by all of the translations of B included in X:

$$X_B = (X \ominus B) \oplus B$$

c. *Close* of X relative to B, denoted by X^B, is the reverse operation to *open*:

$$X^B = (X \oplus B) \ominus B$$

d. *Boundary determination* (δX):

$$\delta X = X \backslash (X \ominus G)$$

The usual structuring element is $G = \begin{matrix} o & o & o \\ o & \mathbf{O} & o \\ o & o & o \end{matrix}$ where $\begin{matrix} o & - & \text{object} \\ \mathbf{O} & - & \text{origin} \end{matrix}$

e. *Thinning*, using morphological transformation, is defined as follows:

$$X \otimes B = X \backslash (X * B)$$

The usual structuring element is $B = \begin{matrix} \mathrm{o} & \mathrm{o} & \mathrm{o} \\ * & \mathbf{O} & * \\ \mathbf{o} & \mathbf{o} & \mathbf{o} \end{matrix}$ where $\begin{matrix} \mathbf{o} & - & \text{object} \\ \mathrm{o} & - & \text{background} \\ \mathbf{O} & - & \text{origin} \\ * & - & \text{don't care} \end{matrix}$

To obtain a simetrical thinning we must apply successively the operation described above, using as structuring element the rotate object B:

$$X \otimes^s B = ((...((X \otimes B^1) \otimes B^2) \otimes ...) \otimes B^n),$$

where $B^1 = B$ and $B^i = Rotate(B^{i-1}), 2 \leq i \leq n$.

f. *Thicking* of X through B, denoted $X \odot B$, is the reverse operation to *thinning* and is defined as follows:

$$X \odot B = X \cup (X * B)$$

g. The *skeleton* of an object X, denoted by $S(X)$, is defined as:

$$S(X) = \bigcup_{n=0}^{n_{max}} s_n(x) = \bigcup_{n=0}^{n_{max}} [(X \ominus nG) \backslash (X \ominus nG)_G],$$

where n_{max} is the smallest n such that $X \ominus nG = \emptyset$.

The reconstructed object X is:

$$X = \bigcup_{n=0}^{n_{max}} [s_n(x) \oplus nG],$$

where $X \ominus nG = (((X \ominus G) \ominus G) \ominus \ldots) \ominus G$.

h. *Prunning* deletes (suppresses) the *parasite branches*, that can be the results after a thinning operation:

$$X_{pn} = X_1 \cup [(X_2 \oplus G) \cap X], \text{ where}$$

$$X_1 = X \otimes^s E;$$

$$X_2 = \bigcup_{j=1}^{8} [X_1 * E^j];$$

$$E = \begin{matrix} * & * & * \\ o & \mathbf{o} & o \\ o & o & o \end{matrix}$$

3. Character Recognition

There are several types of recognitions, and we use syntactical recognition because it has the advantage of being able to identify an infinity set of complex forms using a small number of production rules. Syntactical recognition of a form assumes *the identification of the primitives* that compund the form (these primitives must be easy to recognise) which is achieved with morphological processing, then *syntactical analysis* of the form in order to identify and perphaps obtain its structure, which will be presented in the next paragraph.

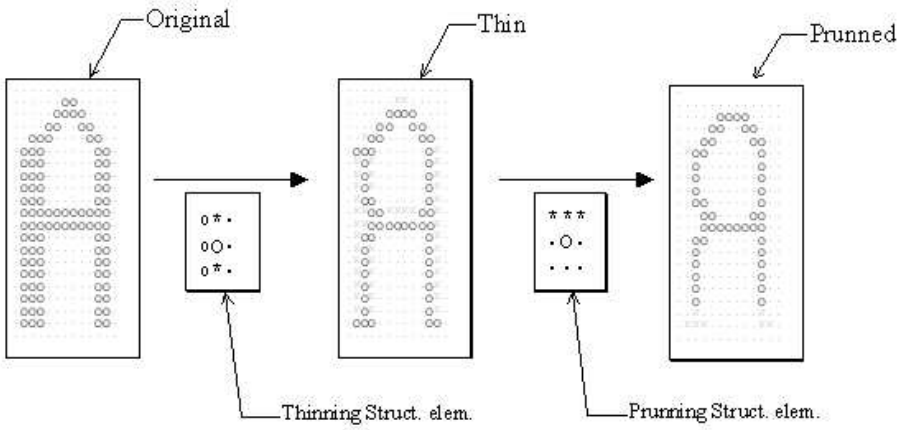

Figure 1. Chracter 'A' during recognitin: original, thinned and prunned

Character recognition can be divided in the following steps, as shown in Figure 1:

a) Thinning, described in Section 2, paragraph e;

Figure 2. The significance of signs from Figure 1

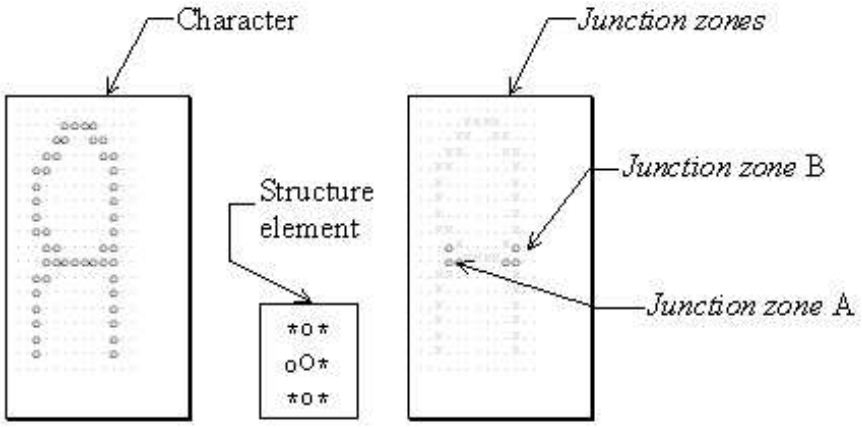

FIGURE 3. The junction zones for character 'A'

b) Then the object is prunned, according to the rules described in Section 2, paragraph h;
c) To obtain the *corners* (Junction zone) we may apply successively the hit-miss operation, using as structuring element the rotate object B, as in 3:

$$B = \begin{matrix} * & \mathbf{o} & * \\ \mathbf{o} & \mathbf{O} & * \\ * & \mathbf{o} & * \end{matrix} \quad \text{where} \quad \begin{matrix} \mathbf{o} & - & \text{object} \\ \mathbf{O} & - & \text{Origin} \\ * & - & \text{don't care} \end{matrix}$$

$$X \otimes B = \bigcup_{i=1}^{4} (X * B^i) \; where: \; B^1 = B \; and \; B^i = Rotate(B^{i-1}),\; 2 \le i \le 4.$$

d) To obtain the *primitives* that compose the image of the character we apply the hit-miss operation with the following structuring elements B:

$$- \; vertical \; lines : X|B = \bigcup_{i=1}^{3} (X \otimes B_i^{|}),\; 1 \le i \le 3,\; with:$$

$B_1^{\|}$			$B_2^{\|}$			$B_3^{\|}$		
*	**o**	*	*	**o**	*	*	**o**	*
o	**O**	*	**o**	**O**	*	**o**	**O**	*
*	**o**	*	*	**o**	*	*	**o**	*

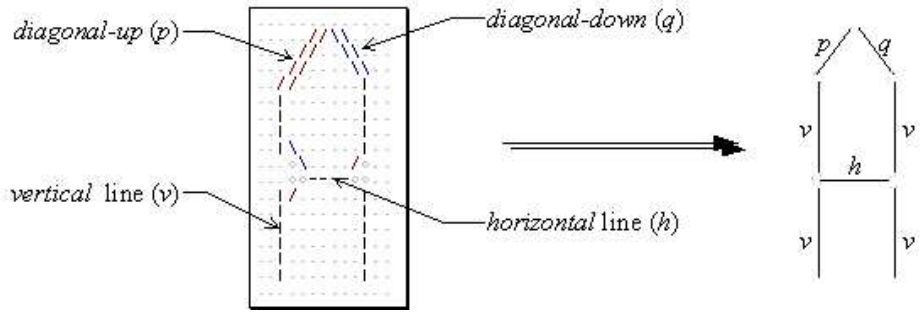

FIGURE 4. The primitives for character 'A'

where the notations have the same significance as in c).

$$- \textit{ horizontal lines} : X - B = \bigcup_{i=1}^{3} (X \otimes B_i^{-}),\ \textit{where}\ B_i^{-} = Rotate(B_i^{|}), 1 \leq i \leq 3$$

$$- \textit{ diagonal} - \textit{down lines} : X \backslash B = \bigcup_{i=1}^{2} (X \otimes B_i^{\backslash}),\ \textit{where}$$

$$B_2^{\backslash} = Rotate^2(B_1^{\backslash}) \text{ and } B_1^{\backslash} = \begin{matrix} * & \mathbf{o} & * \\ * & \mathbf{O} & \mathbf{o} \\ * & * & * \end{matrix}$$

$$- \textit{ diagonal} - \textit{up lines} : X/B = \bigcup_{i=1}^{2} (X \otimes B_i^{/}),\ \textit{where} B_i^{/} = Rotate(B_i^{\backslash}), 1 \leq i \leq 2.$$

Now, it is easy for someone to recognise the primitives that compose the picture and after that to describe the character. It's not difficult to find the relations of relative positions of these primitives (see Figure 4).

4. GRAMAMTICAL DESCRIPTION

A form will be the result of several concatenation operations, that unifies the simplest forms in subforms more and more complex.

A form description language defines their structure, and a form description grammar defines the subform compounding rules.

Structural representations of a form is recommended when there exist complex concatenation relations between the primitives that have been used, because this representation gives us a graphical image of the way in which the primitives are interconnected.

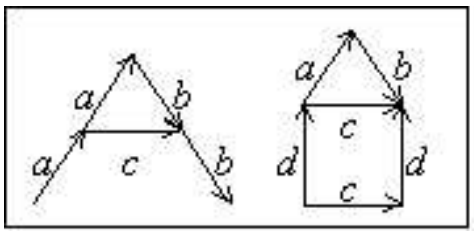

FIGURE 5. Forms built with 'diagonal' primitives

In order to build a form representation, we choose a starting point, which will be the initial node in the structural descrition. This node is then decomposed in two subforms together with the relation between them, as shown in Figure 8. If these subforms are also compound from other subforms, then we will continue the description until all the nodes will contain only primitives.

The syntactical recognition process assumes the following three steps:

- the selection of the primitives in order to obtain the form description alphabet;
- the form representation using a description language;
- the definition of a grammar, such that it will generate the description language.

The first two steps have already been discussed. We will focus next on some grammar classes, which are used for certain form representation methods.

The forms that can be represented through a primitive sequence (using primitive concatenation operation) are called *one-dimension forms*, and their corresponding grammars are called *one-dimension grammars*.

The grammar $G_1 = (\{S, A, C, D\}, \{a, b, c, d, (,), +, *, \neg\}, P, S)$, with the production rules: $P = \{S \to A/C, D \to (a+b)*c, A \to a+D+b, C \to ((\neg d)+c+d)*D\}$, generates the language $L(G_1) = \{a + (a + b) * c + b, ((\neg d) + c + d) * (a + b) * c\}$, where $+$ is the usual concatenation operator, $*$ is for parallel concatenation, and $\neg$ is the operator for extremes inversion [4, 1].

This language describes the two forms from Figure 5, built with the primitives $\{a, b, c, d\} = \{\nearrow, \searrow, \to, \uparrow\}$.

We will present now the image desciption model that uses grammars whose terminal symbols are graphical primitives (horizontal, vertical and diagonal lines, as shown in Figure 4) and predicates [2]. To each primitives we attach characteristics regarding their position in the image frame (framework coordinates). The predicates express relations that can exist between certain image components (primitives and compound forms). The compound forms correspond to the nonterminal symbols from a traditional grammar. For example, $on(x, y)$ ("x is over

y") is a predicat that has the value true if and only if all the components of object x are *on* all the components of object y. The relative positions of the image components can be obtained from their attached coordinates (as we shall see in the next section).

Such a grammar $G = (N, \Sigma, P, S)$ consists of the set of nonterminal symbols $N(S \in N)$, the set Σ of terminal symbols containing primitives and predicates, the set P of production rules (each of them defining a form) and the start symbol S. A production of this grammar has the form $A \rightarrow (\omega) : \alpha; \pi$, where $A \in N$ is a nonterminal representing the name of the defining graphical form, ω is a list of arguments of the production, α is a list of objects (primitives or compound objects), and π is a predicate (α and π are statements that should be true for arguments from the list ω.

In order to ilustrate these concepts, we will use as example the 'A' character from Figure 4. Suppose that the terminal alphabet is:

- types of graphical primitives: $/, \backslash, |, -$;
- predicates: 'in','on','near','right'.

The productions described above can be represented in a parsing tree. For this example, the corresponding tree is shown in Figure 8.

We shall present now an algorithm for *generating graphical subforms and for building the production rules* (corresponding to the first step of the syntactical recognition process).

At the begining, the description list ($ObIni$) contains the graphical primitives from the drawing (we denote by $GPNo$ the total number of graphical primitives), storing for each primitive p its type ($PrimitiveType(p)$), the window in which is framed ($Domain_p = (x_1, y_1, x_2, y_2)$) and the set of components ($MOb_p = \{p\}$). For each graphical primitive from the analized drawing we will store the following information:

$$(p, PrimitiveType(p), Domain_p, MOb_p = \{p\}), \quad (for\ each\ p = 1, 2, ..., GPNo).$$

This list is enriched with new compound objects ($NewOb$) formed from the ones existing in the list $ObIni$ and are in a convenient relative position, denoted by relation ϕ (for example, (s, d), if s is inside d, i.e. $Domain_s \subset Domain_d$). For a compound object o we will store:

$$(o, (s, d), \phi(s, d), Domain_o, MOb_o = MOb_s \cup MOb_d),$$

where $Domain_o$ is computed using the following formula:

$$x_1 := Minim(s.x_1, d.x_1), \quad x_2 := Maxim(s.x_2, d.x_2),$$
$$y_1 := Minim(s.y_1, d.y_1), \quad y_2 := Maxim(s.y_2, d.y_2).$$

(see Figure 6).

The addition of new elements is continued until it is no more possible ($NewOb = \emptyset$).

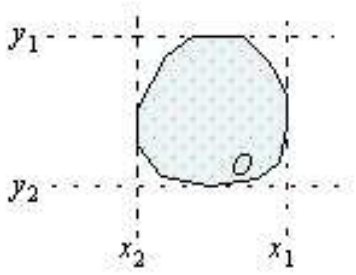

FIGURE 6. Coordinates specification for an object

Those objects o that contain all the graphical primitives of the drawing ($MOb_o \supset \{1, 2, ..., GPNo\}$), represent drawing descriptions.

Algorithm 3.1.

Initialize $ObIni = \{Ob_p, p = 1, GPNo\}$; {List of graphical primitives}
$ObAd := ObIni; LPrim := ObIni; o := m;$
Repeat
$NewOb := \emptyset; LOb := ObIni \cup ObAd;$
For each $\alpha = (s, d) \in LOb \times ObAd \cup ObAd \times LOb$ execute
For each relation ϕ execute { see the Table 1 }
If $\phi(s, d)$ *and* $Ob_s \cap Ob_d = \emptyset$ then {s ϕ d and no common primitives}
$o := o + 1$; *Compute* $Domain_o$;
$NewObi := NewOb \cup \{(o, (s, d), \phi(s, d), Domain_o, MOb_s \cup MOb_d)\}$;
$ObIni := LOb$; $ObAd := NewOb$;
Until $NewOb = \emptyset$; {no more new objects}

The set of descriptions: $= \{o | \{1, 2, ..., GPNo\} \subset MOb_o\}$.

Two objects can be in the relations presented in 1, representing relations between their coordinates (specified in 6).

ϕ	O_1 *in* O_2	O_1 *on* O_2	O_1 *near* O_2	O_1 *left* O_2
	$O_2.x_1 < O_1.x_1$	$O_1.y2 \leq O_2.y_1$	$O_1.x_2 \leq O_2.x_1$	$O_1.x_2 < O_2.x_1$
	$O_1.x_2 < O_2.x_2$	$O_1.y_2 \approx O_2.y_1$	$O_1.x_2 \approx O_2.x_1$	$O_1.x_2 \not\approx O_2.x_1$
	$O_2.y_1 < O_1.y_1$	$O_1.x_1 \approx O_2.x_1$	$O_1.y_1 \approx O_2.y_1$	$O_1.y_1 \approx O_2.y_1$
	$O_1.y_2 < O_2.y_2$	$O_1.x_2 \approx O_2.x_2$	$O_1.y_2 \approx O_2.y_2$	$O_1.y_2 \approx O_2.y_2$

TABLE 1. Definition of relations ϕ, where $\alpha \approx \beta$ means $|\alpha - \beta| \leq \varepsilon$, namely are very close to each other.

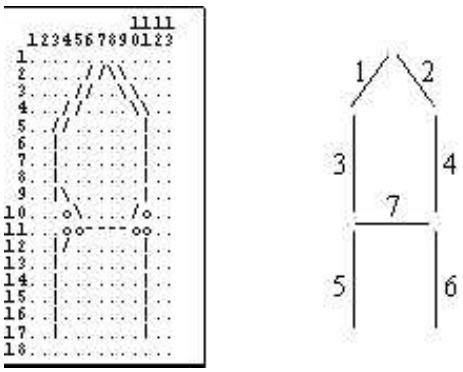

FIGURE 7. The numbers associated to primitives

Example 4.1.
We consider the drawing from Figure 4, and we will number the primitives (as in 7) in order to build the initial subform list, as presented in Table 2.

	Pr.	x_1	y_1	x_2	y_2
1.	/	3	2	7	5
2.	\	8	2	11	4
3.	|	3	6	3	9
4.	|	11	5	11	9
5.	|	3	11	3	16
6.	|	11	11	11	16
7.	-	6	10	9	10

TABLE 2. The primitives used in Example 4.1

Applying the predicates 'in', 'on', 'near' and 'left' to all object pairs from the initial list we will obtain the following list (when we add new elements, we also determine information regarding the new objects postions):

8-(1,2):near(1,2);
9-(3,4):left(3,4);
10-(3,5):on(3,5);
11-(4,6):on(4,6);
12-(5,6):left(5,6)

The process is repeated using the extended list (1-12). The new list elements obtained in the third step are:

13-(8,9):on(8,9);
14-(9,12):on(9,12);
15-(10,11):left(10,11);

The next step will produce the following elements:

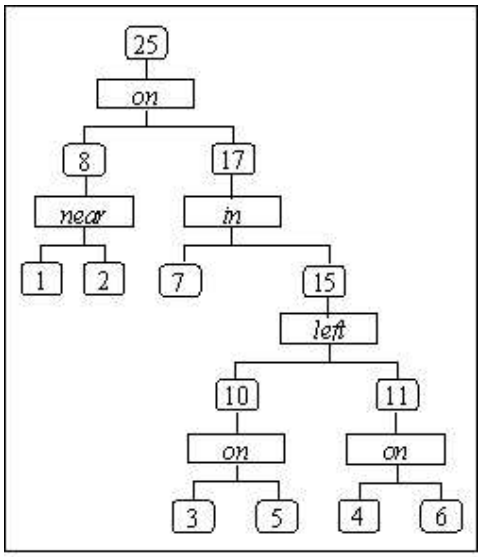

FIGURE 8. The parsing tree for character 'A'

16-(7,14):in(7,14);
17-(7,15):in(7,15);
18-(8,14):on(8,14);
19-(8,15):on(8,15);
20-(12,13):on(13,12);

Since there is no object that contains all the primitives, the process will continue, adding the last elements:

21-(7,18):in(7,18);
22-(7,19):in(7,19);
23-(7,20):in(7,20);
24-(8,16):on(8,16);
25-(8,17):on(8,17).

We notice that these last objects (21–25) contain all the primitives as components. The parsing tree corresponding to the object 25 is shown in Figure 8.

Next, we will replace variables with constant names, in three steps:

1) symbols from 8 to 20, representing intermediary compound objects will be replaced with nonterminals (from A to M):

A=8-(1 near 2)	E=12-(5 left 6)	I=16-(h in G)
B=9-(3 left 4)	F=13-(8 on 9)	J=17-(h in H)
C=10-(3 on 5)	G=14-(9 on 12)	K=18-(A on G)
D=11-(4 on 6)	H=15-(10 left 11)	L=19-(A on H)
		M=20-(F on E)

2) symbols from 1 to 7, representing the primitives will be replaced with the primitive names:

A=8-(p near q)	E=12-(v left v)	I=16-(7 in 14)
B=9-(v left v)	F=13-(A on B)	J=17-(7 in 15)
C=10-(v on v)	G=14-(B on E)	K=18-(8 on 14)
D=11-(v on v)	H=15-(C left D)	L=19-(8 on 15)
		M=20-(13 pe 12)

3) symbols from 21 to 25, representing final objects have the following associated grammar:

21: S → h in K;	22: S → h in L;	23:S → h in M;
K → A on G;	L → A on H;	M → F on E;
A → p near q;	A → p near q;	E → v left v;
G → B on E;	H → C left D;	F → A on B;
B → v left v;	C → v on v ;	A → p near q;
E → v left v.	D → v on v.	B → v left v.

24: S → A on I;	25: S → A on J;
A → p near q;	A → p near q;
I → h in G;	J → h in H;
G → B on E;	H → C left D;
B → v left v;	C → v on v;
E → v left v.	D → v on v.

We will proceed now with the reduction of the grammar, in order to obtain a simpler one. The reduction takes into consideration the following remarks:

a. we have some identical rules: C and D, B and E, respectively;
b. rules for K and L, I and J differ only by a symbol (G, respectively H), and we will use unification;
c. rules for M and F differ only by a symbol (F, respectively A), so we will use unification.

The final grammar, capable of generating the original sample is:

S → h in K | h in M | A on I;	
K → A on G;	A → p near q;
G → B on B | C left C;	B → v left v;
	C → v on v ;
M → M on B | A on B;	I → h in G;

Since the rule for the symbol M generates sequences of the form $A(on\ B)^n, n > 0$, we can extend it even more such that $n \geq 0$, yielding, eventually, to $M \rightarrow M\ on\ B|A$. This grammar generates character of the form shown in 9.

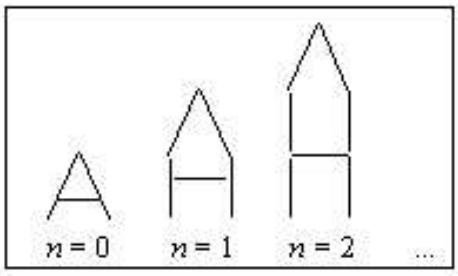

FIGURE 9. Forms obtained for different values of n with the determined grammar

As it can be seen in the figure 9, the proposed method allows us to obtain classes of generalized images, starting from an initial pattern, images that satisfy the same rules.

The ability of understanding an image will be increased if we know the rules under which they were generated (if the images had been defined by relations between components, and not by static or semantic rules). In such cases, the importance of lingvistical methods is obvious, since these methods represent tools useful in solving recognition problems.

The final step in pattern recognition process is the parsing (syntactical analysis), in which the forms are classified according to the description grammars of the form classes. The parsing decides if a certain form belongs or not to a form class, and when the answer is afirmative it will provide information about the form structure.

In our case, a form is a drawing, and a form class is a picture language. The problem of recognising a drawing d (representing the studied form) from a picture language P (a given form class) is: "$d \in P$?". Consequently, we have presented here a method of solving this problem.

REFERENCES

[1] K.S. Fu, *Syntactic Pattern Recognition, Application*, Springer-Verlag, New York, 1977.
[2] E.B. Hunt, *Artificial Intelligence, Pattern Recognition, Grammatical Pattern Classification*, Academic Press, New York, San Francisco, London, 1975, pg. 144-182.
[3] A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, London, 1989.
[4] A.C. Shaw, *A formal picture description scheme as a basis for picture processing system*, Inform. and Control, 14, 9, 1969.
[5] A. Vlaicu, *Prelucrarea digitală a imaginilor*, Editura Albastră, Cluj-Napoca, 1997.
[6] T. Pavlidis, *Algorithms for Graphics and Image Processing*, Springer-Verlag (Berlin, Heildelberg), 1982.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania
E-mail address: per|motogna@cs.ubbcluj.ro