THE ABSTRACT THEORY OF AUTOMATA

THE ABSTRACT THEORY OF AUTOMATA

V.M. GLUSHKOV

Contents

Introduction

The present article is devoted to an exposition of a new field of discrete mathematics, the study of discrete (digital) automata from the point of view of abstract algebra. This field, which we call the abstract theory of automata, occupies a place between the theory of logical systems on the one hand and the general theory of algorithms, on the other. The abstract theory of automata is closely bound up with well-known parts of algebra, especially the theory of semi-groups and can thus be considered as one of the divisions of contemporary abstract algebra.

Notwithstanding the abstractness emphasised in its name, the abstract theory of automata has an extensive field of applications: many results find direct application in the logical design of digital electronic calculating machines and other discrete automata.

Two different forms of the exposition of the abstract theory of automata appear to be possible, emphasising either the applied or the theoretical side. The exposition of the abstract theory of automata in the applied aspect has been given in the author's article [12]. The aim of the present article is to consider the abstract algebraic aspect of the theory, while applications play a lesser part.

It should be emphasised that the great majority of articles in which the abstract theory of automata have been treated are written from the applied point of view. The articles which have been devoted to the abstract algebraical aspect of the theory are un-coordinated and are not unified by a common general principle. The systemization of the material in the field of the abstract-algebraical theory of automata is no easy problem.

The solution of this problem was the basic aim of the seminar in Kiev on the abstract theory of automata directed by the present author. Studies commenced in the autumn of 1959. The present article gives a summary of the work of this seminar.

This article is intended as a survey. The present state of the theory of automata necessitated a thorough revision of the available material and its extension with many new results, not hitherto published. Thus the majority of the theorems, described in this article, appear either quite new or generalizations and modifications of already known results. In the latter case, references are made to the original source.

Apart from the results obtained by the author himself, several new results are cited, by other members of the Kiev seminar, (V.G. Bodnarchuk and A.A. Letichevskii). The material in this article was given as a report by the author at the All-Union Colloquium in general algebra, held at Sverdlovsk in September 1960.

§I. Homomorphism and Equivalence of Automata

The concept of automata, with which the abstract theory of automata operates, is a reasonable abstraction from actually existing discrete (digital) automata on which contemporary computing technique and numerous systems of discrete automatic control and management are based. In implicit form, without being strictly defined, the idea of abstract automata arose when the so-called *sequential* (or non-primitive) computing systems began to be studied.

The work of Huffman [29] started the abstract study of automata and was the first to break away from considerations of practical design and construction. Even in this work, the idea of abstract automata was not quite clearly defined. The work itself has a purely applied character, while its exposition is carried out in the language of relay-contact schemes. Clearer conceptions of the abstract theory of automata were brought out in the work of Mealy [19] and especially in the work of Moore [20]. Mealy and Moore used for their constructions two conceptions of abstract automata different somewhat from each other. Seshu [25] suggested that these two differing types of automata might be distinguished by the names *Mealy* and *Moore*.

To begin, we state the definition of the more general concepts of Mealy's automata; these will now be referred to simply as automata.

DEFINITION 1. By an *automaton* (*Mealy*) we mean an object $A = A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ consisting of three non-empty sets $\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}$ and two functions $\delta = \delta(a, x)$ and $\lambda = \lambda(a, x)$ defined in these sets. $\mathfrak{A}$ is called the *set of states* of the automaton, $\mathfrak{X}$ the *set of input signals* and $\mathfrak{Y}$ the *set of output signals* of automaton A. The function $\delta(a, x)$ is called *transfer function* of the automaton. It gives a map of the set $\mathfrak{A} \times \mathfrak{X}$ to the set A. The function $\lambda = \lambda(a, x)$ gives a map of the set $\mathfrak{A} \times \mathfrak{X}$ to the set $\mathfrak{Y}$ and is called the *output function* of automaton A. Automaton A is called an *initial automaton*, if in the set $\mathfrak{A}$, there exists a certain element a_0, called the *initial state of automaton A*.

The concept of the automaton, introduced in definition 1, differs from the concept given by Mealy, in that the sets $\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}$ need not necessarily be finite and can be chosen arbitrarily. In addition, in definition 1 the difference between *initial* and *non-initial* automata is emphasised. The absence of clearly defined distinctions between these two types of automata in papers on the abstract theory of automata often leads to an undesirable confusion of ideas.

Using definition 1, it is not difficult to define the concept of *homomorphism* and *isomorphism* of automata and the concept of a *sub-automaton* to any given automaton. Whenever an automaton is defined by these three sets $\mathfrak{A}, \mathfrak{X}$ and $\mathfrak{Y}$, it will be possible to introduce several different concepts of homomorphism, sub-automata etc. In introducing these concepts, a special notation will be used showing clearly the dependence of the concepts on the sets $\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}$.

It is not difficult to explain the idea of the notation indicated above by an example of homomorphism of automata. A homomorphism ψ of an automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ onto an automaton $B(\mathfrak{A}_1, \mathfrak{X}_1, \mathfrak{Y}_1, \delta_1, \lambda_1)$ is naturally defined by a set of three mappings: $\psi_1 : \mathfrak{A} \longrightarrow \mathfrak{A}_1$, $\psi_2 : \mathfrak{X} \longrightarrow \mathfrak{X}_1$ and $\psi_3 : \mathfrak{Y} \longrightarrow \mathfrak{Y}_1$, satisfying, for any arbitrary elements $a \in \mathfrak{A}$ and $x \in \mathfrak{X}$ with the supplementary relations $\psi_1(\delta(a, x)) = \delta_1(\psi_1(a), \psi_2(x))$, $\psi_3(\lambda(a, x)) = \lambda_1(\psi_1(a), \psi_2(x))$.

To emphasise that the homomorphism ψ, defined in this way, operates on all three sets $\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}$, we shall call it an $(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y})$ homomorphism.

Other types of homomorphism are possible. It often happens, for example, that in automata A and B both the input and output sets of signals coincide: $\mathfrak{X} = \mathfrak{X}_1$ $\mathfrak{Y} = \mathfrak{Y}_1$. In this case, the homomorphism of automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ onto the automaton $B(\mathfrak{A}_1, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$ is defined by only one mapping ψ: $\mathfrak{A} \longrightarrow \mathfrak{A}_1$, satisfying the supplementary relation $\psi(\delta(a, x)) = \delta_1(\psi(a), x)$, $\lambda(a, x) = \lambda_1(\psi(a), x)$. This type of homomorphism is called an $\mathfrak{A}$*-homomorphism*. $\mathfrak{X}$*-homomorphisms*, $\mathfrak{Y}$*-homomorphisms*, $(\mathfrak{A}, \mathfrak{X})$ *homomorphisms*, $(\mathfrak{A}, \mathfrak{Y})$*-homomorphisms* and $(\mathfrak{X}, \mathfrak{Y})$*-homomorphisms* are defined in a similar way. It is necessary to point out that in the notation used in the definition of homomorphisms, isomorphisms, sub-automata etc. we define once for all the meaning of the symbols $\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}$ for the sets of states of input and output signals of the automaton, no matter in what way these sets may be represented in any concrete automata with which we have to do.

In the definitions of homomorphisms and isomorphisms of *initial* automata it is always assumed (unless stated to the contrary), that the corresponding mappings transform the initial state of the first automaton into the initial state of the second automaton. If it is necessary to distinguish between homomorphisms (isomorphisms) of initial and non-initial automata, we shall sometimes use the terms "initial" and "non-initial" homomorphisms (isomorphisms).

As usual, automata may be defined as *isomorphic* if they are reciprocally related by a one-one homomorphism. Automata A and B are called mutually isomorphic if there exists a reciprocally one-one homomorphism of the one automaton on the other. Corresponding to the various type of homomorphism there will be similar types of isomorphism. Thus, for example, the assertion that the automata A and B show an $\mathfrak{A}$*-isomorphism* denotes that the automata A and B coincide in the sets of input and output signals, and that there is a one to one mapping of the states of the one automaton onto those of the other, which leaves the input and output transitions unaltered.

For short a $(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y})$ homomorphism (isomorphism) will be called a *homomorphism (isomorphism)*. All other cases of homomorphism (isomorphism) can be considered clearly as partial cases of $(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y})$ homomorphism (isomorphism) when one or other of the defined homo(iso)morphisms gives an identical mapping.

An $\mathfrak{A}$ homo(iso)morphism can be called homo(iso)morphisms on states, $\mathfrak{X}$- and $\mathfrak{Y}$- homo(iso)morphisms - homo(iso)morphisms on input and output automata.

Similar methods may be used to define a sub-automaton. The automaton $B(\mathfrak{A}_1, \mathfrak{X}_1, \mathfrak{Y}_1, \delta_1, \lambda_1)$ is called a sub-automaton (more exactly an $(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y})$ sub-automaton) of the automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ if $\mathfrak{A}_1 \subseteq \mathfrak{A}$, $\mathfrak{X}_1 \subseteq \mathfrak{X}$, $\mathfrak{Y}_1 \subseteq \mathfrak{Y}$, and the functions δ_1 and λ_1 coincide with the functions δ and λ in the subsets $\mathfrak{A}_1$ and $\mathfrak{X}_1$. As well as $(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y})$ sub-automata, other types can be visualized. The automaton $B(\mathfrak{A}_1, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$ may be called an A-subautomaton of $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ if $\mathfrak{A}_1 \subseteq \mathfrak{A}$, and the functions δ_1 and λ_1 agree with δ and λ over the corresponding sub-set.

In the case of initial automata we shall distinguish between *initial* and *non-initial* sub-automata. In the first case the sub-automaton B of the initial automaton is an initial automaton and its initial state coincides with the initial state of automaton A. In the second case in which B is a non-initial sub-automaton, and none of its possible states need coincide with the initial state of A.

An automaton is called finite if the three sets which define it viz:- $\mathfrak{A}$, $\mathfrak{X}$, $\mathfrak{Y}$ are finite. $(\mathfrak{X}, \mathfrak{Y})$-finite automata are of special importance in the abstract theory. These automata have finite sets of input and output signals but the set representing the states may be finite or infinite at will. Other types of finite automata can also be defined.

To proceed further with the theory of automata we may associate two free semi-groups $F(\mathfrak{X})$ and $F(\mathfrak{Y})$ with an automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$. These semi-groups are generated by the sets $\mathfrak{X}$ and $\mathfrak{Y}$ of input and output signals and are called the *input and output semi-groups*. The elements of these semi-groups are called the *input and output words* of the automaton A. In the same way sometimes the elements of the set $\mathfrak{X}$ and $\mathfrak{Y}$ may be called *input and output letters* and the sets $\mathfrak{X}$ and $\mathfrak{Y}$ themselves the *input and output alphabets* of the automaton A.

The elements of the semi-groups $F(\mathfrak{X})$ and $F(\mathfrak{Y})$ are formed from finite selections of letters of the input and output alphabets. The numbers of letters in a word is called its *length*. As well as words with positive length $(\geqslant 1)$, it is convenient to associate a null-word (i.e. with zero length) with the semi-groups $F(\mathfrak{X})$ and $F(\mathfrak{Y})$. The fixed letter e is used to denote the null-word.

Together with the input and output semi-groups for an automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ it is desirable to consider the free semi-group $F(\mathfrak{A})$ generated by the set $\mathfrak{A}$ of states of the automaton. Let us agree to call this the *semi-group of states* of A, and its elements *the state-words* of this automaton.

It is now possible to realise a natural extension of the definition of the transfer function $\delta(a, x)$ and the output function $\lambda(a, x)$ of automaton

A from the set $(\mathfrak{A} \times \mathfrak{X})$ to the set $\mathfrak{A} \times F(\mathfrak{X})$ by the following law: for an arbitrary input word $p = x_1x_2 \ldots x_k$ and an arbitrary state $a \in \mathfrak{A}$ we put $\delta(a, p) = a_1a_2 \ldots a_k$ where $a_1 = \delta(a, x_1)$, $a_2 = \delta(a_1, x_2) \ldots a_k = \delta(a_{k-1}, x_k)$ and $\lambda(a, p) = y_1y_2 \ldots y_k$, where $y_1 = \lambda(a, x_1)$, $y_2 = \lambda(a_1, x_2) \ldots y_k = \lambda(a_{k-1}, x_k)$. In the case $p = e$ we assume that $\delta(a, p) = \lambda(a, p) = e$.

The functions $\delta(a, x)$ and $\lambda(a, x)$ obtained in this way giving the mapping of the set $\mathfrak{A} \times F(\mathfrak{X})$ to the set $F(\mathfrak{A})$ and the set $F(\mathfrak{Y})$ respectively will be called the *extended transfer function* and *extended output function* of the given automaton A. The words $a_1a_2 \ldots a_k$ and $y_1y_2 \ldots y_k$ may be called the *state-word* and the *output word*, corresponding to the input word p in state a.

DEFINITION 2. The mapping of the input semi-group into the output semi-group making the output word $\varphi_a(p) = \lambda(a, p)$ from the arbitrary input word p, is called *mapping* φ_a *induced by the state* $a \in \mathfrak{A}$ *of automaton* $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$. The aggregate of all the different mappings $\varphi_a (a \in \mathfrak{A})$ is called the *family of mappings*, induced by the automaton A. The representation, induced by the initial state is called the *mapping induced by the initial automaton.*

DEFINITION 3. Two automata A and B in which both input and output alphabets are the same are called *equivalent*, if they induce the same family of mappings. If A and B are initial automata they are called *initial equivalents*, if their initial conditions induce identical mappings. Finally, the states of one or of several different automata are called *mutually equivalent* if they induce identical mappings. The following theorem is an immediate consequence of definition 3.

THEOREM 1. *Two automata A and B are mutually equivalent if and only if there corresponds to any arbitrary state of A an equivalent state in B and conversely.*

It is easy to see that two $\mathfrak{A}$-isomorphic automata (isomorphic with respect to states) are mutually equivalent (the converse is not true). A more general theorem is the following.

THEOREM 2. *If there exists an $\mathfrak{A}$-homomorphism (state-homomorphism) of an automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ with an automaton $B(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$, then the automata A and B are mutually equivalent. For any state $a \in \mathfrak{A}$ the states a and $\psi(a)$ are mutually equivalent.*

PROOF. By theorem 1 and the proof of theorem 2, it is sufficient, obviously, to prove the equivalence of states a and $\psi(a)$. Let $p = x_1x_2 \ldots x_k$ be any arbitrary input word, and let $\delta(a, p) = a_1a_2 \ldots a_k$ be the corresponding state-word in the state a of A, and let $\delta_1(b, p) = b_1b_2 \ldots b_k$ be the corresponding state-word in state $b = \psi(a)$ of B. By the definition of $\mathfrak{A}$-homomorphism we have:

$$\begin{aligned}
\psi(a_1) &= \psi(\delta(a, x_1)) = \delta_1(b, x_1) = b_1,\\
\psi(a_2) &= \psi(\delta(a_1, x_2)) = \delta_1(\psi(a_2), x_2) = \delta_1(b_1, x_2) = b_2,\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
\psi(a_k) &= \psi(\delta(a_{k-1}, x_k)) = \delta_1(\psi(a_{k-1}), x_k) = \delta_1(b_{k-1}, x_k) = b_k.
\end{aligned}$$

If we now denote, by $q = \lambda(a, p) = y_1y_2 \ldots y_k$, the output word corresponding to the input word p in the state a of the automaton A, and, by

$q' = \lambda_1(b, p) = y_1'y_2' \ldots y'_k$, the output word corresponding to the input word in the state $b = \psi(a)$ of the automaton B, then by the definition of $\mathfrak{A}$-homomorphism we get:

$$y_1 = \lambda(a, x_1) = \lambda_1(\psi(a), x_1) = \lambda_1(b, x_1) = y_1',$$
$$y_2 = \lambda(a_1, x_2) = \lambda_1(\psi(a_1), x_2) = y_2',$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$y_k = \lambda(a_{k-1}, x_k) = \lambda_1(\psi(a_{k-1}), x_k) = \lambda_1(b_{k-1}, x_k) = y_k'.$$

Thus $q = q'$ and consequently, the theorem is proved.

From theorem 2 and the obvious symmetry properties and transitivity for equivalence of automata, we get the following result.

LEMMA. *If, given two automata B and C a third automaton A, can be found, which is $\mathfrak{A}$-homomorphic to B and C, the automata B and C are mutually equivalent.*

A stronger form of this result holds as well.

THEOREM 3. *In the set $\mathfrak{M}$ of all mutually equivalent automata there is one and, up to $\mathfrak{A}$-isomorphism, only one automaton on which any arbitrary automaton of the set $\mathfrak{M}$ can be mapped by an $\mathfrak{A}$ homomorphism. All states of this automaton are non-equivalent in pairs and the power of the set of its states does not exceed the power of the set of states of any automaton of the set $\mathfrak{M}$.*

PROOF. We denote by $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ an arbitrary automaton of the set $\mathfrak{M}$. The set $\mathfrak{A}$ of states of the automaton A can be separated into non-intersecting sub-sets of mutually equivalent states. We shall call these sub-sets the equivalence classes of the automaton A. The set of all such classes will be denoted by $\mathfrak{B}$ and from it we may construct an automaton $B(\mathfrak{B}, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$.

The output function $\lambda_1(b, x)$ of the automaton B may be defined as follows: for any arbitrary state b of B and for any arbitrary input signal $x \in X$, let $\lambda_1(b, x) = \lambda(a, x)$, where a is any arbitrary state of the automaton A, belonging to the equivalence class b. Because of the definition of equivalence of states, the precise meaning of the function $\lambda_1(b, x)$ does not depend on the choice of state a from the equivalence class b.

It follows directly from the definition of the mappings induced by the states of the automaton , that the equivalence of states a_1 and a_2 of A implies that the states $\delta(a_1, x)$ and $\delta(a_2, x)$ will be equivalent. Thus an arbitrary input signal x transforms every state composing the equivalence class b into input states of one and the same equivalence class b_1. Putting $\delta_1(b, x) = b_1$, we define the transfer function of the automaton B.

It is easily seen that any arbitrary state a of A is equivalent to the state b of the automaton B which contains it (the equivalence class of automaton A). Actually, we write $p = x_1x_2 \ldots x_k$, an arbitrary input word of the automata A and B, and let

$$a_1a_2\ldots a_k = \delta(a, p),$$

$$y_1y_2\ldots y_k = \lambda(a, p)$$

(δ and λ are extended transfer and output functions of the automaton A).

We denote by b_i the equivalence class containing the state $a_i (i = 1 \dots k)$ and construct an extended transfer function $(\delta_1(b, p))$ and an extended output function $(\lambda_1(b, p))$ of the automaton B. We shall have

$$\delta_1(b, p) = b'_1 b'_2 \dots b'_k, \quad \lambda_1(b, p) = y'_1 y'_2 \dots y'_k,$$

where

$$b'_1 = \delta_1(b, x_1), \; b'_2 = \delta_1(b'_1, x_2), \; \dots, \; b'_k = \delta_1(b'_{k-1}, x_k);$$
$$y'_1 = \lambda_1(b, x_1), \; y'_2 = \lambda_1(b'_1, x_2), \; \dots, \; y'_k = \lambda'_1(b'_{k-1}, x_k).$$

We denote by ψ the mapping which takes every state of the automaton A to its equivalence class. From the definition of functions δ_1 and λ_1, we get:

$$b'_1 = \delta_1(b, x_1) = \psi(\delta(a, x_1)) = \psi(a_1) = b_1,$$
$$b'_2 = \delta_1(b'_1, x_2) = \delta_1(b_1, x_2) = \psi(\delta(a_1, x_2)) = \psi(a_2) = b_2,$$
$$\dots\dots\dots\dots\dots\dots\dots\dots$$
$$b'_k = \delta_1(b'_{k-1}, x_k) = \delta_1(b_{k-1}, x_k) = \psi(\delta(a_{k-1}, x_k)) = \psi(a_k) = b_k;$$
$$y'_1 = \lambda_1(b, x_1) = \lambda(a, x_1) = y_1,$$
$$y'_2 = \lambda_1(b'_1, x_2) = \lambda_1(b_1, x_2) = \lambda(a_1, x_2) = y_2,$$
$$\dots\dots\dots\dots\dots\dots\dots\dots$$
$$y'_k = \lambda_1(b'_{k-1}, x_k) = \lambda_1(b_{k-1}, x_k) = \lambda(a_{k-1}, x_k) = y_k.$$

Thus the words $y_1 y_2 \dots y_k$ and $y'_1 y'_2 \dots y'_k$ are the same, which because of the arbitrary choice of the word $p = x_1 x_2 \dots x_k$ implies that the states a and $b = \psi(a)$ are equivalent. By theorem 1 this means that automata A and B are mutually equivalent. Thus the automaton B is contained in the set $\mathfrak{M}$.

If there are two mutually equivalent states b' and b'' of the automaton B, then, taking any arbitrary input states a' and a'' of an automaton A contained in them it follows from the previous argument that the state a' is equivalent to the state b', and a'' to b''. Because the equivalence relation satisfies the conditions of symmetry and transitivity it follows that the states a' and a'' are equivalent. Thus these states belong to the same equivalence class, that is, the states b' and b'' coincide. This completes the proof that in the automaton B no two states are equivalent.

Now, let $C(\mathfrak{S}, \mathfrak{X}, \mathfrak{Y}, \delta_2, \lambda_2)$ be an arbitrary automaton belonging to the set $\mathfrak{M}$. As shown above, B and C are equivalent. Using theorem 1, a state b of B can be found equivalent to any arbitrary state c of C. Since B does not contain mutually equivalent states, the state b is uniquely determined by the state c. Thus, assigning to every state c of the automaton C the equivalent state $\varphi(c) = b$ of the automaton B, we can define a mapping φ of the set of states $\mathfrak{S}$ of the automaton C on the set of states $\mathfrak{B}$, of the automaton B. We must now show that the mapping φ is an $\mathfrak{A}$ homomorphism.

Actually, for an arbitrary state c of C and for an arbitrary input signal x, we have $\lambda_2(c, x) = \lambda_1(\varphi(c), x)$, because of the equivalence of C and $\varphi(c)$. Further, since equivalent states are transformed by arbitrary input signals into equivalent states we have $\psi(\delta_2(c, x)) = \delta_1(\psi(c), x)$. Thus the mapping φ is shown to be an $\mathfrak{A}$, homomorphism. Because the mapping is many - one the power of the set $\mathfrak{B}$ of states of the automaton B will not exceed the power of the set $\mathfrak{C}$ of states of the automaton C. Since C may be chosen arbitrarily this proves theorem 3 completely.

The last theorem suggests the following

DEFINITION 4. An automaton (Mealy) is called *reduced*, if any two different states of it are not equivalent.

The proof of theorem 3 is therefore essentially the method of constructing the *reduced* automaton (Mealy) equivalent to the automaton A. This method is a generalisation for arbitrary automata (Mealy) of the method of reduction for finite automata developed originally by Mealy (19) and extended later by Aufenkamp and Hohn (1), Netherwood (21), Paul and Unger [22] (cf. [12]). The developments referred to describe principally the methods of the actual determination of equivalence classes for finite automata and lead to the same idea.

This idea may be stated as follows ([12]):- to begin with all states of the automaton are arranged in so-called 1-classes. States a, b, ... are put in the same 1-class, if the output signals $\lambda(a, x)$, $\lambda(b, x)$ ... are the same for any arbitrary choice of the input signal x. 1-classes can be further separated into 2-classes. The states a, b, ... contained in the same 1-class, are assigned to the same 2-class, if for any arbitrary choice of input signal x the states $\delta(a, x)$, $\delta(b, x)$ are found the same 1-class $K(x)$, depending on the choice of x, but not the choice of states a, b, ... The operation of *splitting* of 1-classes into 2-classes can be carried over without change to the 2-classes. From the splitting of 2-classes 3-classes arise: applying the same operation to them, we get 4-classes and so on. In the case of finite automata the $(i + 1)$th class will be coincident with the ith class for a finite value i, $i \geqslant 1$. The i-classes in this case are the equivalence classes of the given automaton.

In practical applications of the abstract theory of automata, we often have cases in which the transfer functions $\delta(a, x)$ and the output functions $\lambda(a, x)$ are defined only for some and not for all pairs of values of a and x. Such automata may be called *partial automata*. The theory of partial automata differs from the theory of ordinary automata (*fully defined automata*) in that the states of partial automata only induce partial and not fully defined mappings. Then the concepts of equivalence of automata and their states can be replaced with the more general concept of *compatability* based on coincidence of induced mappings in their intersections of their fields of definition [2].

The reduction theory for partial automata is much more complicated than that for fully defined automata, because the theorem corresponding to theorem 3 is not true for partial automata. Because of lack of space this theory cannot be set out in detail here[1]. For practical purposes an adequate method in most cases is to extend the definition of the operations so as to make the partial into an ordinary automaton and to consider the reduction of this ordinary automaton.

A series of special methods of reduction of finite partial automata has been developed by Aufenkamp [2], Netherwood [21], Paul and Unger [22] Ginsburg [8]. All these methods are developed from the point of view of

[1] Those interested can find the details in [12].

applications and will not be discussed here.

Because of the possibility of extending the definitions of partial automata mentioned above, every partial automaton can be considered as a partial sub-automaton of some fully-defined automata. For the purposes of the present article, it is enough to mention such a possibility and to limit the following to the examination of fully-defined automata only. The term "automaton" always means the normal, fully-defined automaton.

As well as the automata already defined (Mealy) we next examine a particular case of such automata, which we shall call a *Moore* automaton.

DEFINITION 5. An automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}), \delta, \lambda)$ is called a Moore automaton if its output function $\lambda(a, x)$ can be represented in the form $\lambda(a, x) = \mu(\delta(a, x))$ where $\delta(a, x)$ is the transfer function of A, and $\mu(a)$ is a univalent function giving a mapping of the set of states $\mathfrak{A}$ onto the set $\mathfrak{Y}$ of its output signals. The function $\mu(a)$ is called the location function or output shift function of A. The value of function $\mu(a)$ for the state a may be called the location of this state.

The concept of this type of automaton (1) for finite automata was first examined by Moore [20] (see also Medvedev [17]). The Moore automata can be considered as a special case of Mealy's type of automata only on the level of abstract theory. When it comes to practical questions of design and discussion of the sequential operations involved, these two cases of automata must be considered as essentially different.

Within the abstract theory the Moore automata are worthy of special study. In some cases the specific properties of Moore automata allow a more natural and basic theory to be constructed than in the case of an arbitrary Mealy automata. From the point of view of the manifold of mappings induced by them, the Moore automata are themselves no poorer than arbitrary Mealy automata.

THEOREM 4. *For any Mealy automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}), \delta, \lambda)$ there exists an equivalent Moore automaton B, moreover if A is finite, B can also be chosen to be finite, with a number of states equal to $(m + 1)n$,* where *m is the number of input signals and n the number of states in A.*

PROOF. To each state a of automaton A we place in correspondence a set of states $(a_0, a_\alpha = (a, x_\alpha))$, where x_α runs through the set of input signals $\mathfrak{X}$. All states obtained in this way are taken to be states of automaton B. We define the transfer function δ_1 of B as follows: $\delta_1(a_0, x_\beta) = (a, x_\beta)$, $\delta_1((a, x_\alpha), x_\beta) = \delta((a, x_\alpha), x_\beta)$. The output function λ_1 of B is defined by the following relations: $\lambda_1(a_0, x_\beta) = \lambda(a, x_\beta)$, $\lambda_1((a, x_\alpha), x_\beta) = \lambda(\delta(a, x_\alpha), x_\beta)$. We denote by $\mu(b)$ the function putting into correspondence with state $b = (a, x_\beta)$ of automaton B the output signal $\lambda(a, x_\beta)$. Then $\lambda_1(a_0, x_\beta) = \mu(\delta_1(a_0, x_\beta))$, $\lambda_1((a, x_\alpha)x_\beta) = \mu(\delta_1(a, x_\alpha), x_\beta)$.

Therefore automaton B is an automaton of type Moore.

We denote, by $a_1a_2 \ldots a_k$ the state word and by $y_1y_2 \ldots y_k$ the output word corresponding to the state a of automaton A for an arbitrary input word $p = x_1x_2 \ldots x_k$. From the definition of the functions δ_1 and λ_1, state a_0 of automaton B makes correspond to the input word p the state-word $b_1b_2 \ldots b_k$, the elements (letters) of which are defined as follows:

$$b_1 = \delta_1(a_0, x_1) = (a, x_1),$$
$$b_2 = \delta_1(b_1, x_2) = (\delta(a, x_1), x_2) = (a_1, x_2),$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$b_k = \delta_1(b_{k-1}, x_k) = (\delta(a_{k-2}, x_{k-1}), x_k) = (a_{k-1}, x_k).$$

In the same way we define the output word $y_1'y_2' \dots y_k'$ which the state a_0 of automaton B makes correspond to the input word p:

$$y_1' = \lambda_1(a_0, x_1) = \lambda(a, x_1) = y_1,$$
$$y_2' = \lambda_1(b_1, x_2) = \lambda_1((a, x_1), x_2) = \lambda(\delta(a, x_1), x_2) = \lambda(a_1, x_2) = y_2,$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$y_k' = \lambda_1(b_{k-1}, x_k) = \lambda_1((a_{k-2}, x_{k-1}), x_k) = \lambda(\delta(a_{k-2}, x_{k-1}), x_k) = \lambda(a_{k-1}, x_k) = y_k.$$

Thus the output words $y_1, y_2, \dots y_k$ and $y_2', y_2' \dots y_k'$ are the same. Since the choice of the word p is arbitrary, this proves that the states a and a_0 are equivalent. Since with any input signal equivalent states are transformed to equivalent states, for any $x_\alpha \in \mathfrak{X}$ the state (a, x_α) of B is equivalent to the state $\delta(a, x_\alpha)$ of A. In this way, there is a state of A equivalent to any state of B. The converse is also true. Hence, by theorem 1, automata A and B are equivalent.

The assertion of *theorem 4 concerning the number of states of* B follows at once from the method of constructing B. This completes the proof of the theorem. This result is a more powerful form of the result obtained by Blokh (4, theorem 1).

In considering operations on Moore automata, one needs to bear in mind that while any sub-automaton of a Moore automaton and any automaton isomorphic to a Moore automaton is a Moore automaton, the property of being a Moore automaton is not in general preserved in homomorphism. In the same way an automaton, equivalent to an automaton of type Moore need not be of this type.

It is necessary therefore to introduce special concepts of Moore *homomorphism* and *Moore equivalence*.

From definition 5, the location function $\mu(a)$ of a Moore automaton is defined in a unique way by its transfer and output functions for all states a which are values of the transfer functions. For the remaining states, the location function is not defined. We will call such states *unlocalisable states*. In the sequel we shall only consider Moore automata in which all states are localisable, by extending the definition of the location function if necessary so that this is attained.

In the further discussion of Moore automata, we shall consider not transfer and output functions, but transfer and location functions, defining, if necessary, the output functions by means of the superposition of the transfer and location functions $\lambda(a, x) = \mu(\delta(a, x))$.

We can now introduce the definition of Moore homomorphism and Moore equivalence.

DEFINITION 6. A *homomorphism* $\varphi = (\varphi_1, \varphi_2, \varphi_3)$ *of a Moore automaton* $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ *on a Moore automaton* $B(\mathfrak{A}_1, \mathfrak{X}_1, \mathfrak{Y}_1, \delta_1, \mu_1)$ is called a Moore homomorphism if the location function is preserved; i.e., for any arbitrary state $a \in \mathfrak{A}$, we have $\varphi_3(\mu(a)) = \mu_1(\varphi_1(a))$, in particular for the case of $\mathfrak{A}$ -homomorphism $\psi(\mu(a)) = \mu_1(\psi(a))$. Equivalent states are called *Moore equivalent*, if they have the same locations. Two Moore automata are called *Moore equivalent* if to any arbitrary state of one there corresponds a Moore equivalent state of the second and vice versa. Theorems analogous to theorems 2 and 3, can now be established for Moore homomorphism and Moore equivalence in the same way as those were proved.

THEOREM 5. *If there exists a $\mathfrak{A}$-homomorphism ψ of an automaton A on an automaton B, then A and B are Moore equivalent. For any arbitrary state a, the states a and $\psi(a)$ are Moore equivalent.*

THEOREM 6. *In the set $\mathfrak{M}$ of all Moore equivalent Moore automata, there exists one and, up to Moore $\mathfrak{A}$-isomorphism, only one, automaton, on which any automaton of the set $\mathfrak{M}$ is mapped in a Moore $\mathfrak{A}$-homomorphism. All the states of this automaton are Moore non-equivalent in pairs, and the power of the set of its states does not exceed the power of the set of states of any automaton from the set $\mathfrak{M}$.*

The proofs of theorems 5 and 6 are almost literal repetitions of theorems 2 and 3.

DEFINITION 7. A Moore automaton is called a *reduced Moore automaton*, if any arbitrary pair of its states are Moore mutually non-equivalent.

The operation of *Moore reduction* is defined by analogy with the general operation of reduction. The difference consists in the determination of Moore equivalence classes: it is necessary to begin not with the 1-classes, but with the construction of the 0-classes, associating in one and the same 0-class all states of a given automaton which have the same location.

We may consider further means of specifying automata. Finite Mealy automata can be fully defined by double entry tables of transfers and output which define respectively the transfer and output functions. The lines of these tables denote input signals, and the columns - the corresponding states of the automaton.

Finite Moore automata are generally defined by so-called *located transfer tables*; these are the usual transfer tables with the locations of the states written at the top of each column.

A more pictorial method of describing automata is that of linear directed graphs. The states of the automaton are shown on the graph by small circles. If the states a and b are connected by the relation $b = \delta(a, x)$ the circle representing the state a on the graph is joined to that representing b, by an arrow marked x. If the output signal $y = \lambda(a, x)$ corresponds to the pair (a, x), then, in the case of Mealy automata, the arrow from a to $b = \delta(a, x)$, is also marked with the output signal y. The arrows joining an ordered pair of states are normally replaced by one arrow.

Another method of notation is used in the case of Moore automata: in the circles, denoting the states of the automaton, the locations of the states are written together with the states themselves. In figures 1 and 2 diagrams are given of two Moore automata (states are denoted by the letters a and b, input signals by x, y and output signals by u, v).

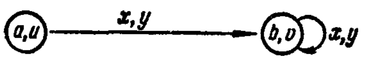

Fig. 1.

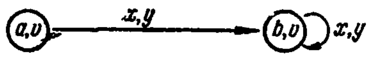

Fig. 2..

It is easy to verify that the automata given by these graphs are isomorphic, but at the same time they are not Moore mutually isomorphic. The transfer tables with locations for these automata are shown in the diagram.

	u	v
	a	b
x	b	b
y	b	b

	v	v
	a	b
x	b	b
y	b	b

The example shows that the concept of isomorphism of Moore automata, regarded as a partial case of Mealy automata, is a more general concept than the concept of Moore isomorphism.

§2. The representation of mappings in automata

Free semi-groups play a large role in what follows. $F(\mathfrak{X})$, $F(\mathfrak{Y})$ etc. denote the free semi-groups with a unit element considered as a set of words in the alphabets $\mathfrak{X}$, $\mathfrak{Y}$ etc. The case of infinite alphabets, which are the free generators of the corresponding infinite semi-groups is not excluded. The semi-groups $F(\mathfrak{X})$, $F(\mathfrak{Y})$ etc. are called semigroups in alphabets $\mathfrak{X}$, $\mathfrak{Y}$ etc.

The mapping of the free semi-group $F(\mathfrak{X})$ on to the free semi-group $F(\mathfrak{Y})$ is said to be represented in the automaton A, if it can be induced by some states of this automaton. By the definition of a mapping induced by the states of an automaton, it follows that not every mapping φ of a free semi-group $F(\mathfrak{X})$ on to a free semi-group $F(\mathfrak{Y})$ can be represented in the automaton. Two necessary conditions emerge from this definition, which the mapping φ must satisfy.

C O N D I T I O N 1. Any arbitrary word p of the free semi-group $F(\mathfrak{X})$ in which the mapping φ is defined, has the same length as its image $\varphi(p)$ in this mapping.

C O N D I T I O N 2. The condition $\varphi(pq) = \varphi(p)r$, where $r = \psi(pq)$ defines a certain word of the free semi-group $F(\mathfrak{Y})$, can be satisfied for any pair of words p and q of the free semi-group $F(\mathfrak{X})$.

D E F I N I T I O N 8. The mapping φ of the free semi-group $F(\mathfrak{X})$ on to the free semi-group $F(\mathfrak{Y})$ is called an *automaton mapping* if the conditions 1 and 2 given above are satisfied. These conditions are called the conditions for the automatonness of φ. $F(\mathfrak{X})$ is called the *input* and $F(\mathfrak{Y})$ the *output* of the semi-group mapping φ, while $\mathfrak{X}$ and $\mathfrak{Y}$ are the *input and output alphabets* of this mapping.

Automaton mappings under various names (recursive sequential functions, sequential operators etc.) have been considered by many authors ([24], [4], [12]).

It appears that the conditions of automatonness of a mapping formulated above are not only necessary but also sufficient conditions for the possibility of the mapping by an automaton. To prove this, it is simplest to use the idea of the free automaton. The sources of this concept are to be found in the work of Huffman [29]. The definition of a free automaton was formulated explicitly by Sorkin [38]. The definition given here differs from Sorkin's definition only in that it refers to transfer as well as to output functions.

DEFINITION 9. Let $\mathfrak{B}$, $\mathfrak{X}$ and $\mathfrak{Y}$ be arbitrary non-empty sets, $F(\mathfrak{X})$ the free semi-group with a unit element generated by the set $\mathfrak{X}$, $\mathfrak{A}$ the set of all words of the form bp, where $b \in \mathfrak{B}$, $p \in F(\mathfrak{X})$. The Moore automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \mu)$ is called a *free automaton*, if its transfer function $\delta(a, x)$ is defined for any arbitrary $a = bp \in \mathfrak{A}$ and any arbitrary $x \in \mathfrak{X}$ with the condition that $\delta(bp; x) = bpx$. The location function $\mu(a)$ can be arbitrary. The set $\mathfrak{B}$ is called the set of free generators of the automaton A.

THEOREM 7. *Any set of automaton mappings $\varphi_\alpha (\alpha \in M)$ with a general input semi-group $F(\mathfrak{X})$ and with a general output semi-group $F(\mathfrak{Y})$ can be represented in a free Moore automaton by the set of free representations which are found to be in one-one correspondence with the set of all $\varphi_\alpha (\alpha \in M)$.*

PROOF. Let us construct a Moore automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \mu)$, with a set of states consisting of all possible words of the form $\varphi_\alpha p$, where p is an arbitrary word of the semi-group $F(\mathfrak{X})$, including the null-word e, while the mapping symbol φ_α is considered simply as a certain letter. For any state $a = \varphi_\alpha p$ of A and any letter x of its input alphabet $\mathfrak{X}$, we define the value of the transfer function $\delta(a, x)$ of A by the relation $\delta(\varphi_\alpha p, x) = \varphi_\alpha px$. The value of the location function $\mu(a)$ for state $a = \varphi_\alpha p$ is taken as the last letter of the word $\varphi_\alpha^{-1}(p)$ in the output semi-group $F(\mathfrak{Y})$ for any non-empty word p and equal to any random letter of the output alphabet $\mathfrak{Y}$ when p is the null-word ($\varphi_\alpha (p)$ here denotes the image of the word p in the mapping φ_α). Let $q = x_1x_2 \ldots x_k$ be an arbitrary (non-empty) word of the input semi-group $F(\mathfrak{X})$, while $r = y_1y_2 \ldots y_k$ is the image of this word in the mapping φ_α. By the construction above the state $\varphi_\alpha e$ of A makes the input word q in the state $\varphi_\alpha e$ correspond to the state word $\varphi_\alpha x_1 \varphi_\alpha x_1x_2 \varphi_\alpha x_1x_2x_3 \ldots \varphi_\alpha x_1x_2 \ldots x_k$.

The output word, corresponding to the word q in the state $\varphi_\alpha e$, is represented as $\mu(\varphi_\alpha x_1)\mu(\varphi_\alpha x_1x_2) \ldots \mu(\varphi_\alpha x_1x_2 \ldots x_k)$ or, by the definition of function μ, in the form $y_1'y_2', \ldots y_k'$ where y_i' is the last letter of the word $\varphi_\alpha (x_1, x_2 \ldots x_i)$ $(i = 1, \ldots, k)$. By the second condition of automatonness $\varphi_\alpha (x_1x_2 \ldots x_i) = y_1y_2 \ldots y_i$, whence $y_1'y_2' \ldots y_k' = y_1y_2 \ldots y_k$.

In this way, the state $\varphi_\alpha e$ of A puts into correspondence any arbitrary input word p with the image of this word in the mapping φ_α. Thus it is proved that in the automaton A all given mappings φ_α $(\alpha \in M)$ can be represented by means of the states $\varphi_\alpha e$.

The representation of automaton mappings in the case of free automata is not always very economical from the point of view of the number of states used. This number is always infinite, although a given mapping can occur in some cases of representation in finite automata. The number of states of the automaton, used in the representation of given mappings can be decreased by method given by Raney [24].

Let us examine this method in the case of only one automaton mapping φ. Let $F(\mathfrak{X})$ be the input free semi-group of the mapping φ, and $F(\mathfrak{Y})$ the output semi-group. For any arbitrary word p from $F(\mathfrak{X})$ we construct a mapping $\varphi_p : F(\mathfrak{X})$ to $F(\mathfrak{Y})$, defining its value $\varphi_p(q)$ for any arbitrary input word q as follows. Because φ is an automaton mapping we have $\varphi(pq) = \varphi(p)r$, where r is a certain output word, the length of which is equal to the length of the word q. We assume then, that $\varphi_p(q) = r$. It is easy to see that all mappings φ_p obtained in this way will be automaton mappings. Raney proposes that all mappings φ_p (including the mapping φ itself, which can be considered as the mapping φ_p for the null-word e) should be called *states of the mapping* φ. Using these concepts, we can prove the following proposition.

THEOREM 8. *Any automaton mapping φ can be represented in a Mealy automaton, the set of states of which correspond with the set of states of the mapping φ.*

PROOF. Let $\mathfrak{A}$ denote the set of all states φ_p of the mapping φ (p is an arbitrary word of the input semi-group $F(\mathfrak{X})$ of mapping φ). Let $F(\mathfrak{Y})$ denote the output semi-group of the mapping φ; we thus define a Mealy automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \mu)$. We define the transfer function of this automaton by the relation $\delta(\varphi_p, x) = \varphi_{px}$, and the corresponding output function by relation $\lambda(\varphi_p, x) = \varphi_p(x)$, where x is any arbitrary letter of the input alphabet $\mathfrak{X}$.

We show that the mapping, induced by state $\varphi = \varphi_e$ (e the null-word) coincides with the mapping φ, given according to theorem 8. Let $p = x_1x_2 \dots x_k$ be an arbitrary input word. The state φ_e of the automaton A associates the word p with the state word $\varphi_{x_1}\varphi_{x_1x_2} \dots \varphi_{x_1x_2\dots x_k}$. If $\varphi(p)$ denotes the image of the word p in the mapping φ and is given by $y_1y_2 \dots y_k$, then by the definition of states of a mapping we shall have $\varphi_e(x_1x_2 \dots x_k) = y_1y_2 \dots y_k$, $\varphi_{x_1}(x_2x_3 \dots x_k) = y_2y_3 \dots y_i$, ..., $\varphi_{x_1x_2 \dots x_{k-1}}(x_k) = y_i$.

The state φ_e associates the word p with the output word

$$q = \lambda(\varphi_e, x_1)\lambda(\varphi_{x_1}, x_2)\dots\lambda(\varphi_{x_1x_2\dots x_{k-1}}, x_k).$$

By the definition of $\lambda(a, x)$ and by the second condition of automatonness for the mappings φ_e, φ_{x_1}, ... $\varphi_{x_1x_2 \dots x_{k-1}}$ we find that $q = y_1y_2 \dots y_k$. In view of the arbitrary choice of the word p, this means that the mapping induced by the state φ_e of A, coincides with the mapping φ, which is therefore represented in the automaton A. This proves the theorem.

Though everything necessary for the proof of theorem 8 is to be found in Raney's work [24], not only is the theorem not proved, but it is not even stated. However, from a practical point of view, the value of theorem 8 is not great, for there are no sufficiently effective direct methods of finding the states of a given automaton mapping. (Indirect methods of solving this problem practically are given by methods developed in the following section). On the theoretical plane, theorem 8 presents undoubted interest in that it shows that there exists a representation of the automaton mapping having the minimum possible number of states.

THEOREM 9. *The power of the set of states of any automaton, in which the given automaton mapping φ is represented is not less than the power of the set of states of this mapping.*

PROOF. Let us consider an arbitrary automaton A, in which the mapping φ is induced by a certain state A. For any arbitrary input word p of A (corresponding at the same time to the input word of the mapping φ) we denote by a_p the last state of the state word, which the state a associates to the input word p. The mapping φ_p, induced by the state a_p, is defined as follows: for any arbitrary input word q the image $\varphi_p(q)$ of this word coincides with the word r in the expression $\varphi(pq) = \varphi(p)r$. All the states of mapping φ can be defined in this way. Thus, associating the states a_p of the automaton A with their induced mappings, we obtain a one valued mapping of a certain part of the set of states of the automaton (possibly coinciding with the whole of this set) on to the set of states of the mapping φ. This proves theorem 9.

As shown above, not every mapping φ of the free semi-group $F(\mathfrak{X})$ on to the free semi-group $F(\mathfrak{Y})$ is found to be an automaton mapping. There is however a simple method (see [12]), by which we can construct a new mapping ψ, which will be an automaton mapping and from which it is possible in a univalent manner to construct the initial mapping φ.

This method consists in adding two new elements (one in each set) to the generating sets $\mathfrak{X}$ and $\mathfrak{Y}$. These can be called *null letters* and denoted by α and β. The mapping ψ is constructed as follows: if p is any arbitrary word from $F(\mathfrak{X})$, and q its image in the mapping φ, then the mapping ψ is defined first for the word $p_1 = p\alpha\alpha \ldots \alpha$, where the number of null letters added on the right equals the length of the word q. The image $q = \psi(p_1)$ of this word in the mapping ψ is $q_1 = \beta\beta\ldots\beta, q$, where the number of null letters β equals the length of the word p.

After defining the mapping ψ on all words of this form, we may define it for all initial sub-words (the word l is called an initial sub-word of the word l, if there is a word l_2 such that $l = l_1 l_2$). We associate each initial sub-word p_2 of the word p_1 with the initial sub-word q_2 of the word $q_1 = \psi(p_1)$ having the same length as p_2. No inconsistency arises in setting up this correspondence. Actually, if the word p_2 contains even only one null letter, then it can only appear in only one word p_1. If the word p_2 does not contain one null letter, then by the definition of the mapping ψ, $\psi(p_2)$, its image consists of a number of null letters, whether it enters as initial sub-word into p_1, or in some other word. If we include the null-word e (containing no letters at all) in the scope of the mapping ψ, and $\psi(e) = e$ we can represent any arbitrary word l, composed of letters of the alphabet $\mathfrak{X}$ and the letter α, in the form $l = l_1 r_1$, where l_1 is a word of the maximum length, for which the mapping ψ can already be defined. Putting $\psi(l) = \psi(l_1)\beta\beta\ldots\beta$ where the number of letters β is equal to the length of word r_1, we obviously get an automaton mapping, mapping the free semi-group $F(\mathfrak{X}')$ into the free semi-group $F(\mathfrak{Y}')$. $\mathfrak{X}'$ means the set of free generators (the alphabet of the semi-group $F(\mathfrak{X}')$), consisting of set $\mathfrak{X}$ supplemented by the null letter α. A similar definition can be given for $\mathfrak{Y}'$ (the alphabet of the semi-group $F(\mathfrak{Y}')$) which is y is supplemented by the null letter β.

It is easy to see that the mapping φ can be obtained from the mapping

ψ in a natural way. If p is an arbitrary word in the alphabet $\mathfrak{X}$, then adding on the right a sufficient number of null letters α, we shall eventually obtain a word p_1, the image of which in the mapping ψ ends with the null letter β. Suppressing all null letters in the word $\psi(p_1)$, we turn it into the word q, which by the method of constructing the mapping ψ, corresponds to the word p in the mapping $\varphi : \varphi(p) = q$.

THEOREM 10. *If φ is an arbitrary mapping of a free semi-group in the alphabet $\mathfrak{X}$ on to a free semi-group in the alphabet $\mathfrak{Y}$, then there is an automaton mapping ψ of the free semi-group in the alphabet $\mathfrak{X} \cup (\alpha)$ on to the free semi-group in the alphabet $\mathfrak{Y} \cup (\beta)$, which is sufficient to define uniquely the initial mapping φ: here α and β are elements, not contained in $\mathfrak{X}$ and $\mathfrak{Y}$.*

The above method of construction of the mapping ψ from the mapping φ, is called the *standard method of equalisation of word-lengths*. It is not always the most economical method of transforming a given mapping φ into an automaton mapping. This can be shown by taking the case when φ itself is an automaton mapping. In practice, it is usual to add sufficient null letters on the right of input words and on the left of output words to ensure that the conditions of automaton mapping are always satisfied. The proof of theorem 10 guarantees that sooner or later an automaton mapping can be found.

It is often possible (in the case when the original mapping φ is not defined for all words) to make use of letters of the original alphabets $\mathfrak{X}$ and $\mathfrak{Y}$ instead of the null letters α and β.

The importance of theorem 10 is that it points the way of including the theory of arbitrary mappings in the theory of automaton mappings and particularly the theory of arbitrary algorithms in the theory of algorithms realizable by automata, which may be called *automaton algorithms*. Many interesting problems arise in this connection, in particular that of finding effective means of deciding whether a given (for example, normal) algorithm is an automaton algorithm.

The problem of constructing the theory of algorithms on the basis of that of automaton algorithms is important because an automaton algorithm can be defined by giving a finite splitting of the free semi-group, the alphabet of which coincides with the alphabet of the given algorithm.

We now consider the more general problem of the representation of arbitrary automaton mappings by the process of subdivision of free semi-groups.

Let φ be an arbitrary automaton mapping, mapping the free semi-group $F(\mathfrak{X})$ with an alphabet $\mathfrak{X}$ on to the free semi-group $F(\mathfrak{Y})$ with an alphabet $\mathfrak{Y}$. For any arbitrary element (letter) y from the set $\mathfrak{Y}$ let S_y denote the set of all words p of the semi-group $F(\mathfrak{X})$, whose images in the mapping φ finish with the letter y. If any of the sets S_y is empty, then by the second condition of automaton mapping, the corresponding letter y does not enter into any word appearing as an image in the mapping φ. We can thus exclude this letter from the generators of the semi-group $F(\mathfrak{Y})$. Making all such exclusions we can suppose without loss of generality that all the sets S_y are not empty.

Clearly the sets S_y do not intersect in pairs, and their union coincides with the whole semi-group $F(\mathfrak{X})$ excluding the null-word e. The given mapping φ uniquely defines the system of subsets $S_y (y \in \mathfrak{Y})$. The converse proposition is that *the system of sets* $S_y (y \in \mathfrak{Y})$ *defines the original mapping* φ *uniquely*.

Let $p = x_1 x_2 \ldots x_k$ by an arbitrary word of the input semi-group $F(\mathfrak{X})$ of the mapping φ. If the word $x_1 x_2 \ldots x_i$ belongs to the set S_{y_i} $(i = 1, 2, \ldots,)$, by the second condition of automaton mapping and by the method of defining the sets S_y the representation of the word $x_1 x_2 \ldots x_k$ in the mapping φ must coincide with the word $y_1 y_2 \ldots y_k$, defined solely using the system of sets S_y $(y \in M)$, without using the mapping φ. When the indices of the sets S_y are not given explicitly, the system of sets S_y $(y \in \mathfrak{Y})$ defines the mapping φ correct by up to the notation for the letters of the output alphabet.

DEFINITION 10. Any set of elements of the free semi-group $F(\mathfrak{X})$ in an alphabet $\mathfrak{X}$ (not necessarily finite) are called *events in the alphabet* $\mathfrak{X}$. *An automaton system of events* in the alphabet $\mathfrak{X}$ is a system of events in this alphabet, non-intersecting in pairs, the set theoretical union of which coincides (excluding the null-word) with the whole semi-group $F(\mathfrak{X})$. The system M of events in an alphabet $\mathfrak{X}$ is called a *canonical system of events of an automaton mapping* φ *with the input semi-group* $F(\mathfrak{X})$ *and the output semi-group* $F(\mathfrak{Y})$, if any event S of the system M consists of all the words of semi-group $F(\mathfrak{X})$, the images of which in the mapping φ end with a given letter (depending on the event S) of alphabet $\mathfrak{Y}$.

THEOREM 11. *The canonical system of events of an automaton mapping* φ *of the input (free) semi-group* $F(\mathfrak{X})$ *is an automaton system of events in the alphabet* $\mathfrak{X}$. *Conversely, each automaton system* M *of events in an alphabet* $\mathfrak{X}$ *can be considered as the canonical system of events of a certain automaton mapping* φ *of the input semi-group* $F(\mathfrak{X})$. *Here the mapping* φ *is defined by the system* M *uniquely, up to changes of notation for the letters of the alphabet of the output (free) semi-group of the mapping* φ.

Theorem 11 reduces the study of automaton mappings to the study of automaton systems of events. In particular, the problem of representation of mappings is essentially reduced to the problem of representation of events in automata, based on the following definition.

DEFINITION 11. *The event* S *in an alphabet* $\mathfrak{X}$ *is said to be represented in the initial automaton* A *by the output signal (letter)* y, if the input alphabet of A coincides with $\mathfrak{X}$, and the set S consists of just those words in the alphabet $\mathfrak{X}$, which are transformed by the mapping induced by the automaton A into the output words, ending with the letter y. Similarly the *automaton system* M *of events* is said to be *represented in the initial automaton* A, if each event of the system is represented in A by any of its output signals.

The problem of the representation of events in an automaton was first discussed by Kleene [14] for finite automata of a special type (nerve-nets). The same problem is considered in a more abstract form by Medvedev [17]. The representation of events by output signals has been considered by the present author [12]. By theorem 11, every automaton system M can be considered as a canonical system of events of a certain automaton mapping φ. Representing the mapping φ by the initial state of an initial automaton, the system of events M is represented (by definitions 10 and 11) by this automaton.

Any event S is an alphabet $\mathfrak{X}$ may be included in an automaton system M of events defined by the same alphabet. For example, it is possible to choose the system that contains, as well as the event S, the event R consisting of all the non-empty words in the alphabet $\mathfrak{X}$, which do not enter into the event S. (If the event R is empty, then the event S itself constitutes the automaton system). In representing the system M in the automaton, we also represent the event S by some output signal.

From the above, combined with theorem 7, we can formulate the following theorem.

THEOREM 12. *Any event can be represented as an output signal of a certain (generally infinite) initial Moore automaton. Any arbitrary automaton system of events can be represented by an initial Moore automaton.*

We observe that it is necessary in proving theorem 12, by using theorem 7, to select a suitable initial sub-automaton belonging to the free Moore automaton considered in theorem 7.

For practical purposes there is special interest in the problem of the representation of events in finite automata. The following section is devoted to this problem. In conclusion we observe that each sub-automaton of a free automaton is itself a free automaton. This was first shown by Sorkin [38].

§3. The representation of events in finite automata. Operations on events

In the previous section, we considered the problem of the representation of events by output signals. In the case of a Moore automaton in which the output signal coincides with the location of a state, a somewhat different approach to the representation of events is more convenient viz:- the representation of events by sets of states of automata.

Let A be an arbitrary automaton, and p an arbitrary non-empty word in its input alphabet. We shall say that the automaton A is transformed by the input word p from a state a to a state b, if b is the last letter of the state word generated by the state a and the input word p (see section 1). Let ap denote the state to which the automaton is transformed from the state a by the input word p. If $p = x_1x_2 \dots x_k$, then the state $a_k = ap$ may be defined by the relations
$a_1 = \delta(a, x_1)$, $a_2 = \delta(a_1, x_2)$ $\dots$, $a_k = \delta(a_{k-1}, x_k)$, where $\delta(a, x)$ is the transfer function of the automaton A. We say that the automaton is transformed by the null-word e from any state a to the same state.

In this and the following section we shall only consider initial automata, i.e. automata with a fixed initial state. The input word p transforms the initial automaton into the state a_0p, where by a_0 is defined as the initial state of this automaton.

DEFINITION 12. We say that the event S in an alphabet $\mathfrak{X}$ *is represented in the initial automaton A by the set M of its states*, if the input alphabet of automaton A coincides with $\mathfrak{X}$, and the event S consists of just those words which transform the automaton from the initial state into the states belonging to the set M.

The above definition agrees with that given by Medvedev [17]. In the case of Moore automata this definition is equivalent that given in definition 11.

Suppose the event S is represented in a certain initial Moore automaton A by the output signal y. Let M denote the set of all the states of A located by the output signal y. If $p = x_1x_2 \ldots x_k$ is any arbitrary word in the input alphabet of the automaton, $a_1a_2 \ldots a_k$ – a state-word, and $y_1y_2 \ldots y_k$ – the output word, corresponding to the word p in the initial state of A, then, by the definition of the location function $\mu(a)$, $\mu(a_k) = y_k$. Thus the non-empty input word p belongs to the event S, if, and only if, it transforms the automaton from the initial state into one of the states of the set M. If the set M contains the original state of the automaton, then the event S' representing it must contain the null-word, though it cannot appear in the event S. Thus, the events S and S' coincide apart from the null-word.

Conversely, let the event S be represented in the initial automaton by the set of states M. Suppose the value of the location function of A to be y_1 for all the states of the set M and equal to y_2 for all the remaining states, consider the event S', represented in A by the output signal y_1. As before, we find that a non-empty input word p occurs in the event S', if and only if it occurs in the event S. If the set M itself includes the initial state, then the event S contains an empty word, which cannot, of course, enter into the state S'. In this way, events S and S' coincide apart from the empty word.

THEOREM 13. *If the event S is represented in the initial Moore automaton A by the output signal y, then this event, supplemented, perhaps, by the null-word, is represented in that automaton by a set of all the states, located by the output signal y. If the event S is represented in the initial automaton A by the set M of states, then, if the location function of A is defined in such a way that it has the value y for all states of the set M and only for such states, we obtain an initial Moore automaton, in which the event S (apart from the null-word if it occurs in S) is represented by the output signal y.*

This theorem allows one to restrict the consideration of the representation of events to that by sets of states. In this treatment the output functions of the automaton do not play any part. Thus, throughout the present section we shall only consider initial automata, with only transfer functions given and without output signals.

In the previous section it was proved that in infinite (in particular, free) automata, any arbitrary events can be represented. The case of finite automata is different. From the simple set theoretical considerations, it is clear that events must exist which cannot be represented in finite automata.

Indeed for any fixed finite non-empty alphabet $\mathfrak{X}$, there exists an ennumerable set of finite initial automata non-isomorphic as regards states, with the alphabet $\mathfrak{X}$ as the set of input signals. Any such automaton can only represent a finite set of events. Therefore the set of all events in an alphabet $\mathfrak{X}$, represented in a finite automaton, must be an enumerable set. At the same time, the set of all events in the alphabet $\mathfrak{X}$ has the power of the continuum. Hence, events must exist in an alphabet $\mathfrak{X}$, which cannot be represented in finite automata, since the power of the set of all these events is that of the continuum.

The first constructive example of a set of events not representable in any finite automaton, was given by Kleene [14]. Such an example is given by the events in any non-empty alphabet, consisting of the set of all the words whose length is the square of an integer. For our present purpose it is more useful to consider another example.

THEOREM 14. *The event S in a two letter alphabet $\mathfrak{X} = (x, y)$ which is composed of all those and only those words, which contain the same number of the letters x and y; cannot be represented in any finite automaton.*

PROOF. Let us suppose that the event S is represented in an initial automaton A with the initial state a_0 by the set M of states of the automaton. Consider the states $a_1 = a_0x$, $a_2 = a_1x$ ..., $a_n = a_{n-1}x$. We see that for any arbitrary $n = 1, 2, \ldots$ these states are all different. If this were not so, we could find two words p_k and p_m having different lengths k and m not containing any letter y's but the states a_0p_k and a_0p_m would coincide. Let q_k be the word of length k, consisting only of letters y. Then the state $a_0p_kq_k$ must be contained in the set M, but the state $a_0p_mq_k$ cannot be contained in that set. On the other hand, since the states a_0p_k and a_0p_m are the same, the states $a_0p_kq_k$ and $a_0p_mq_k$ must coincide. This contradiction establishes the theorem.

One of the most important problems in the abstract theory of automata is the problem of formulating a language for the description of events, represented in a particular class of automata. The first version of such a language was given by Kleene [14]. This language, which we shall call the language of regular notation was further developed by Copy, Elgot and Wright [15] and by the present author [9].

Another version of a language for the description of events, represented in finite automata was suggested by Medvedev [17]. By using special systems of coding the input alphabets it is possible to construct a third language, based on the use of the language of the calculus of predicates of the second kind for special or limited predicates (finite sets of natural numbers). This version has been considered by Trakhtenbrot [28] and Büchi [5].

We shall employ the language of regular notation which is more easily adapted to the abstract algebraical point of view which we have adopted. Practical methods of synthesis of an automaton based on this language are given in the author's articles [9] and [12].

The language of regular notation makes use of three operations on events, called disjunction, multiplication and iteration. The first two of these operations are two-place operations and the third is a unary operation.

All these operations are defined for events in the same alphabet. If need be the alphabet could be extended, since an event in a smaller alphabet can always be considered as an event in any larger alphabet.

DEFINITION 13. The *disjunction* $S_1 \cup S_2$ of two events S_1 and S_2, is defined as the set theoretical union of these events. The event consisting of all words of the form p_1p_2 where $p_1 \in S_1$, $p_2 \in S_2$, is called the *product* S_1S_2 of the events S_1 and S_2. The event consisting of the empty word and all words of the form $p_1p_2 \ldots p_k$, where $p_1, p_2, \ldots, p_k$ are words of events S and $k = 1, 2, \ldots$ is called the *iteration* $\{S\}$ of the event S.

DEFINITION 14. The set of all events in an alphabet $\mathfrak{X}$ together with the operations of disjunction, multiplication and iteration is called the *algebra of events*. The event, consisting of all words in an alphabet $\mathfrak{X}$, (including the empty word), is called the *most general event* in the alphabet $\mathfrak{X}$. The event consisting of the empty set of words is the *impossible event*. Finally, the events each of which consists of a one letter word $x \in X$, and the event consisting of the empty word e, are called *elementary events* in the alphabet $\mathfrak{X}$.

Finally let us introduce the concept of a regular event, which is one of the central concepts of the theory of finite automata. As stated above, this concept is the same as that introduced by Kleene [14], although it differs greatly from it in form.

DEFINITION 15. Any event in a finite alphabet $\mathfrak{X}$, which can be constructed from elementary events in this alphabet using a finite number of disjunctions, multiplications and iterations, and also the impossible event, is called a *regular event*. Any representation of a regular event in terms of elementary events using the operations of the algebra of events in the alphabet $\mathfrak{X}$ is called a *regular expression* of this event. The symbol ϕ will be used to denote the regular expression of the impossible event.

A regular event can have many different normal expressions, since the algebra of events contains a series of relations leading to identical transformations of regular expressions, i..e. such transformations as leave the events which these expressions represent unaltered.

The relations establishing the associativity and the commutativity of disjunction and the associativity of multiplication belong to the identical relations in the algebra of events. It is not difficult to show that multiplication is distributive with respect to disjunction.

For writing expressions in the algebra of events, in particular regular expressions, one must agree on a natural order for carrying out operations of the algebra. Other things being equal, the operations of iteration are carried out first, then those of multiplication and lastly those of disjunction. Every deviation from this natural order is denoted by round brackets, which we will call common, as opposed to curly brackets, which are used exclusively to denote iteration and are therefore called iteration brackets.

The associativity of disjunction and multiplication allows us to write down disjunctions and products of any finite number of events without using brackets.

An elementary (one element) event will be identified with the element composing it. The null-word will always be denoted by the letter e.

Using these symbols and abbreviations we may write down several relations in the algebra of events for arbitrary events S, R and of the impossible event ϕ:

$$\{\{S\}\} = \{S\}, \tag{1}$$

$$\{S\} = e \vee S\{S\}, \tag{2}$$

$$S\{S\} = \{S\}S, \tag{3}$$

$$\{\{S\}\{R\}\} = \{S \vee R\}, \tag{4}$$

$$\{e\} = e, \tag{5}$$

$$Se = eS = S, \tag{6}$$

$$S\phi = \phi S = \phi, \tag{7}$$

$$S \vee \phi = S, \tag{8}$$

$$\{\phi\} = e. \tag{9}$$

To these relations must be added the relations of associativity, commutativity and distributivity referred to above. The problem of finding a complete system of relations in the algebra of events is an interesting one in the case when the alphabet contains more than one letter, i.e. a system which allows the establishment of the identity of any two arbitrary expressions in the algebra of events. We shall show later that the problem of establishing the identity of regular expressions can be solved algorithmically.

We shall show, first of all, that all events represented in finite automata are regular and we shall construct algorithms making it possible to find a regular expression for the events represented in any finite initial automaton by any set of its states. This result was first proved for finite automata by Kleene [14], although his method was very ineffective for practical applications. The present author (see [10] and [12]) constructed an algorithm suitable for the actual determination of regular expressions for events, represented in finite automata (called the algorithm of analysis of finite automata). We do not state this algorithm, but the algorithm constructed by McNaughton and Jamanda [16]. Although this last algorithm is less useful for practical purposes, the underlying theory is simpler than that of the author. We notice that in contrast to [16], which used a two-letter input alphabet, we can construct an algorithm for any arbitrary finite alphabet.

THEOREM 15. *All events represented in finite automata, are regular. An algorithm exists which allows us to construct a regular expression for events, represented in any finite initial automaton by any set of its states*[1].

PROOF. It is sufficient to construct an algorithm, allowing us to determine a regular expression for an event, represented in any finite initial automaton by any one of its arbitrary states. Actually, the event represented by a set of states of a finite automaton, coincides with the disjunction of the events represented by the states comprised in the set. Consider the arbitrary finite automaton A, the sets of which are denoted by the sequence of integers 1,2 ... n. Let i and j be any two of

[1] This automaton is defined by tables of its transfer and output functions.

its arbitrary states, and let $p = x_1x_2 \ldots x_k$ be the input word transforming the automaton A from the state i to the state j. We may say that the word p transfers the automaton from state i to state $j = ip$ through the *intermediate* states $ix_1, ix_1x_2, \ldots, ix_1x_2 \ldots x_k$.

We denote by $S_{ij}^k (i, j = 1, \ldots, n; k = 0, 1, \ldots, n)$ the event consisting of those non-empty words, which transfer the automaton from the state i to the state j, excluding as intermediate states all states other than the states $1, 2, \ldots, k$. In particular, S_{ij}^0 denotes the event transferring automaton A from state i to state j directly without passing through any intermediate states.

By the definition of the automaton, each of the events S_{kj}^0 represents either an impossible event, or a letter of the input alphabet or the disjunction of such letters (finite because A is a finite automaton A). In this way, each of the events S_{kj}^0 is regular, and the regular expressions for all such events in terms of the transfer function of A can be found. Let us suppose that all the events $S_{ij}^0, S_{ij}^1, \ldots, S_{ij}^{k-1}$ have been proved regular and that the algorithm for constructing their regular representations exists. It is easily seen that for any arbitrary states i and j S_{ij}^k can be constructed by the following recursion formula.

$$S_{ij}^k = S_{ij}^{k-1} \vee S_{ik}^{k-1} \{S_{kk}^{k-1}\} S_{kj}^{k-1} \quad (i, j = 1, \ldots, n). \tag{10}$$

In fact, inspection of the above formula shows that the event R, defined by the right hand side of (10), consists only of those non-empty words which transfer the automaton A from state i to state j, using only the states $1, 2, \ldots, k$ as intermediate states. In this way, $R \subset S_{ij}^k$. Conversely, let p be an arbitrary non-empty input word, transferring A from the state i to the state j and using as intermediate states only the states $1, 2, \ldots, k$, i.e. it is an arbitrary word of the event S_{ij}^k. If the state k is not among the intermediate states, then the word p enters into event S_{ij}^{k-1}, and consequently into the event R. If the state k is an intermediate state, the word p can be represented in the form $p = p_1p_2 \ldots p_m$ $(m \geqslant 2)$, where the word p_1 transfers A from the state i into the state k, using as intermediate states only states $1, \ldots, k-1$, each of the words $p_2, \ldots, p_{m-1}$ transfers the automaton from the state k into the state k, using as intermediate states only the states $1, \ldots, k-1$, and the word p_m transfers the automaton from the state k into the state j, also using only states $1, 2, \ldots, k-1$ as intermediate states. In other words $p_1 \in S_{ik}^{k-1}$, $p_2 \in S_{kk}^{k-1}, \ldots, p_{m-1} \in S_{kk}^{k-1}$, $p_m \in S_{kj}^{k-1}$. Thus by the definition of iteration $p \in S_{ik}^{k-1} S_{kk}^{k-1} S_{kj}^{k-1} \subset R$ (when $m = 2$ the value of the iteration bracket is e).

If p is an arbitrary word from S_{ij}^k, we have shown that $S_{ij}^k \subset R$. Together with the converse inclusion, which has already been proved, it gives $S_{ij}^k = R$, hence the recursion formula (10) is proved. By induction with respect to k, we can show that all the events $S_{ij}^k (i, j = 1, \ldots, n)$ are normal, and the normal expressions for them are obtained by the same method (the continued use of the formula (10)). We observe that if any of the events on the right hand side of (10) are impossible events, then we must use formulae (7)-(9) to obtain the normal expressions.

The event S^n_{ij} consists of all non-empty words, transferring A from the state i into the state j. If $j \neq i$, then S^n_{ij} is the event represented in A by the state j in the initial state i. If then $j = i$, then S^n_{ij} is obviously equivalent to the event $S^n_{ij} \vee e$. This completes the proof of theorem 15.

NOTE. The fact that null-word e enters explicitly into the representations of the events S^k_{ij} in the proof of theorem 15 proves at the same time (by theorem 13) that events represented by the output signal in any finite Moore automaton and consequently, by theorem 4 in any finite Mealy automaton, are regular.

Having established the existence of the analysis algorithm of finite automata, which allows us to determine the normal expressions for events represented in these automata, we turn to the construction of the converse algorithm, which is called the *synthesis algorithm.*

The general synthesis algorithm, described by the present author [9], combines the advantages of generality with convenience in practical application. In contrast to other methods ([14]-[17], [23], [28]) the present method operates with any finite system of normal events, and does not make use of a special coding-system of letters of the input alphabet. It also makes it possible to restrict the number of states required in any automaton to a relatively low value. With a number of additional improvements, it approaches the ideal minimum synthesis of an automaton. For further information on this point see [12].

THEOREM 16. *Any regular event can be represented in a finite initial automaton by some set of its states. There is an algorithm which enables one, given any finite system M of regular events in an alphabet $\mathfrak{X}$, defined by regular expressions, to construct a finite initial automaton which represents each event of the system M by some set of its states. The number of states in this automaton does not exceed $2^n + 1$, where n is the number of different letters of the alphabet $\mathfrak{X}$ which appear in the expressions of the set M.*

PROOF. It is sufficient to prove only the second part of the theorem, as the first part follows immediately from the second. Let us consider the complex K of regular expressions, representing the events of the given system M, and let us attach a number to all letters of the basic alphabet $\mathfrak{X}$ entering into these expressions (the null-word e excluded). In this every appearance of a particular letter receives its particular number. The order in which the numbers are assigned does not matter. For example, in the complex K, constructed out of two regular expressions $R = x\{y\}$ and $S = xx \vee \{y\}x\{y\}$ the following numbers may be assigned to the letters: $R = x_1\{y_2\}x_3$, $S = x_4x_5 \vee \{y_6\}x_7\{y_8\}$. The order of operations in the algebra of events defined above fixes a definite sequence of letters in converting any regular expression into a word. Thus, in the expression R given above, after the letter x_1 either the letter y_2 or the letter x_3 may follow, and similarly after the letter y_2 either the letter x_3 or the letter y_2 may follow. Initial letters may also be defined, i.e. those letters which come first in the conversion of a regular expression into a word. In the expressions above, R and S, the initial letters will be x_1, x_4, y_6 and x_7.

We also define final letters of regular expressions, i.e. those letters such that when they are written down in the correct order of conversion (beginning with the initial letter) of a regular expression they end the regular event represented by this expression. For the regular expression R only x_3 is a final letter, while, for S, x_5, x_7 and x_8 are final letters.

We shall construct an initial automaton A, all states of which, except the initial state a_0, will represent some sets (including, generally, the null-set) of sequentially numbered letters, which appear in the regular expressions of a given complex K. The transfer function $\delta(a, x) = ax$ of this automaton is defined in the following way. Corresponding to any arbitrary letter x of the basic alphabet $\mathfrak{X}$, the state a_0x is chosen to be the set of all sequentially numbered letters x, which appear as initial letters in the complex K.

For any arbitrary set M of sequentially numbered letters of the complex K, considered as a state of the automaton A, we define the state Mx as the set of all those sequentially numbered letters x which follow at least one of the letters of the set M. Only those sets of letters are selected which are obtained by continued application of the rule described above, starting from sets of the form a_0x.

From the definition of regular expressions and the method of constructing the transfer function of the automaton A, if follows that a non-empty word p in the alphabet $\mathfrak{X}$ belongs to an event Q', represented by some regular expression Q of the complex K, if and only if the set a_0p contains just one final letter of the expression Q. The equation $a_0p = a_0$ can only hold when p is the null-word. Thus the event Q' is represented in the automaton just constructed by the set of all those states, which contain at least one final letter of the expression Q, and if the event Q' contains the empty word e, by the original state a_0 as well. The number of the states of the automaton constructed is finite.

In this way the required algorithm is constructed, and theorem 16 is proved. A value for the number of states is easily obtained, if we remember that the number of different sets, which can be constructed out of n (numbered) letters is 2^n.

To clarify the operation of this algorithm, we shall construct a minimal automaton A, representing the events of complex K_1, as described in the proof of theorem 16. The basic alphabet $\mathfrak{X}$ consists in this case of the two letters x and y. Writing the initial state of the automaton A and using the notation of theorem 16, we obtain the following relations: $a_0x = a_1$, $a_0y = a_2$, where a_1 is the set consisting of letters x_1, x_4, x_7, and a_2 is the set consisting of one letter y_8. Further, we have $a_1x = a_3$, $a_1y = a_4$, $a_2x = a_5$, $a_2y = a_2$, where $a_3 = (x_3, x_5)$, $a_4 = (y_2, y_8)$, $a_5 = (x_7)$; $a_3x = a_5$, $a_3y = a_6$, $a_4x = a_7$, $a_4y = a_4$, $a_5x_6 = a_6$, $a_5y = a_8$, where a_6 is the empty set of letters, $a_7 = (x_3)$, $a_8 = (y_8)$; $a_6x = a_6$, $a_6y = a_6$, $a_7y = a_6$, $a_7x = a_6$, $a_8x = a_6$, $a_8y = a_8$.

These relations define completely the transfer function for the automaton A. The automaton A has nine states and represents the event R by the set of states a_3 and a_7, and the event S by the set of states a_1, a_3, a_4, a_5 and a_8. In figure 3 the structure of the automaton is shown graphically. Side by side with the circles, denoting the states which represent one or both of the events R and S, the symbols of the corresponding events are placed.

We notice, that in the present example the location function can be introduced by assigning the states a_0, a_2, a_8 to one output signal, the state a_3 to another, the state a_7 to a third, and the states a_1, a_4, a_5 and a_8 to a fourth output signal. This converts the given automaton into an initial Moore automaton. The event R is then represented by the second and third output signals and the event S by the third and fourth output signals. By constructing the reduced Moore automaton, Moore equivalent to the above automaton according to theorem 6, we can minimize the number of states with conservation of the property of representing the initial complex of events. These considerations are not limited only to the present example but have general validity. Applying minimization to the above example the number of states can be reduced from nine to eight.

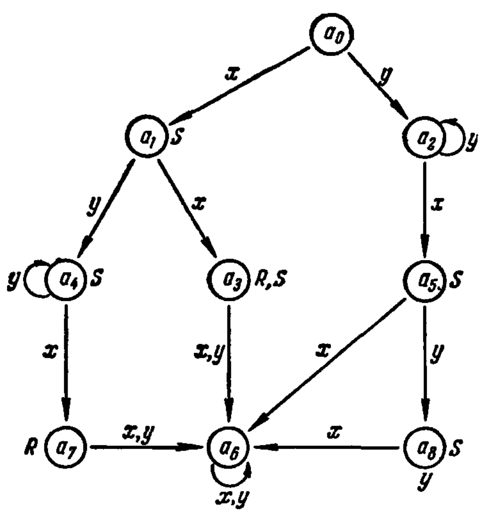

Fig. 3.

From theorem 16 a series of propositions important in the theory of finite automata and the algebra of events can be deduced.

THEOREM 17. *An algorithm exists, which allows us to determine for any arbitrary pair of regular expressions, whether the events represented by these expressions coincide or not.*

PROOF. The required algorithm coincides with the synthesis algorithm described in theorem 16. On constructing a finite initial automaton, representing the events R and S, corresponding to the given expressions by the sets M and N of states, the question of coincidence or non-coincidence of the events R and S is the same as the coincidence or non-coincidence of the finite sets M and N.

We now define further operations on events. The intersection of events is defined as usual in set theory. The collection of all words in an alphabet $\mathfrak{X}$ (not excluding the null-word), which are not contained in an event S is called the complement $\overline{S}$ of the event S in the alphabet $\mathfrak{X}$. The aggregate of all initial segments of all words of the event S, including these words themselves and the empty letter e is called the supplement of the event S.

In setting up the synthesis algorithm, as in theorem 16, for the initial complex K of normal expressions of events, representations of all the unions (disjunctions), all the intersections of these events as well as the complement and supplement of any of them will be represented.

In fact, the union and intersection of the representative sets represent the union and intersection of the corresponding events. The complement (in the set of all states of the automaton) of the set of states, representing any arbitrary event of complex K, represents the complement of this event. It is not so simple as regards the supplement. From the algorithm, of theorem 16, it is easy to see that the supplement of an event defined by any arbitrary regular expression S of the initial complex K, is represented by the set, consisting of the initial state, and of all those states (sets of numbered letters) which contain at least one (numbered) letter entering into the expression S.

Combining this result with that of theorem 15, we have :-

THEOREM 18. *The complement and supplement of any arbitrary regular event, and also the intersection of any arbitrary finite set of regular events are regular events. There exists an algorithm, which allows us, for any arbitrary event, defined by a regular expression, to determine the regular expressions for the complement and supplement of this event. An algorithm also exists allowing us, for any arbitrary finite number of events, defined by regular expressions, to determine a regular expression for the intersection of all these events.*

The operations defining the complement and the intersection of events have been considered in many articles ([9], [14], [16], [17]). Kleene [14] and Medvedyev [17] describe two other operations on events.

The first operation,which we call the extension of the event, is defined as follows. The event S in alphabet $\mathfrak{X}$, containing at least one non-empty word, is transformed into the event $S'F \vee e$, where F is the general event in the alphabet $\mathfrak{X}$, and S' is the event S without the empty word e. The extension of the event which consists of the empty-word itself is the empty-word. If the general event contains the empty word, the extension of any arbitrary event will include the empty word. Medvedyev [17] showed that the extension of an event, represented in a finite automaton, is also represented in the finite automaton. We shall not prove Medvedyev's result, which is a simple consequence of the following theorem thanks to theorem 16.

THEOREM 19. *The general event in a finite alphabet and the extension of any regular event are regular events. A simple algorithm exists permitting us to write down this regular event and similarly to write down a regular expression for its extension.*

If the alphabet $\mathfrak{X}$ consists of a finite number of letters $x_1, x_2, \dots, x_n$, then the general event F in alphabet $\mathfrak{X}$ is represented by a regular expression $\{x_1 \vee x_2 \vee \dots \vee x_n\}$. If the event S is regular, the alphabet of this event is finite and, consequently, the expression $S'F \bigcup e$ of the event S can be written down as a regular expression (from theorem 18, the exclusion of the empty word from a regular event leads to a regular event).

The second operation, called the contraction of an event, is dual to the operation of the extension of an event. In the extension of the event S all words are included, which contain the words of the event S as initial segments. The contraction of the event S is composed of all those and only those non-empty words, which themselves enter into S and all non-empty initial segments of which are also contained in event S.

Denoting by $C(S)$ the contraction and by $R(S)$ the extension of an arbitrary event S, we may write (see [17]):

$$C(S) = \overline{R(\overline{S})}. \tag{11}$$

The bar over the S denotes the complement of S.

Formula (11) is essentially the same as the well-known statement in the restricted calculus of predicates relating up the quantifiers of generalization and existence. The only difference is in the necessity to take into account the special role of the empty word.

Formula (11) together with theorems 18 and 19, leads to the following result.

THEOREM 20. *The contraction of any arbitrary regular event is a regular event. An algorithm exists allowing us for any arbitrary event, given by a regular expression, to obtain a regular expression for the contraction of this event.*

Medvedyev [17] considers another operation on events (substitution). We shall examine not this, but its generalization suggested by Bodnarchuk. This is the superposition of events.

Consider the arbitrary event $S = S(y_1, y_2, \ldots, y_m)$ in the finite alphabet $\mathfrak{Y}$, consisting of the letters $y_1, y_2, \ldots, y_m$, and m arbitrary events $R_1, \ldots, R_m$ in a finite alphabet $\mathfrak{X}$, consisting of letters $x_1, x_2, \ldots, x_n$. The superposition $S(= R_1, R_2, \ldots, R_m)$ of the events $R_1, \ldots, R_m$ and of the event S is defined to be the event in the alphabet $\mathfrak{X}$ which consists of all those words which are obtained by replacing each letter y_{i_s} in any word $y_{i_1}y_{i_2} \ldots y_{i_k}$ of the event S by any arbitrary event R_{i_s}.

THEOREM 21. *The superposition of regular events $R_1, \ldots, R_m$ and of the regular event S in the alphabet $\mathfrak{Y} = \mathfrak{Y}$ $(y_1, y_2, \ldots, y_m)$ is itself a regular event. The regular expression for such a superposition is obtained by replacing the letters $y_1, \ldots, y_m$ in the regular expression of the event S by regular expressions defining the events $R_1, \ldots, R_m$: the regular expression for the event R_i replaces every appearance of y_i in the regular expression of the event S $(i = 1, 2, \ldots, m)$.*

Bodnarchuk has devised a direct method of constructing a finite initial automaton representing the superposition of events $R_1, \ldots, R_m$ and of the event S directly from the automaton representing these events. It is easy to see that we can construct a new synthesis algorithm for automata, representing given regular events, starting from automata, which represent elementary events and regular events.

In Medvedev's paper [17], he used a different language for representing events different from the language of regular expressions.

Translated into the language used in the present article this method is as follows:- in any arbitrary given finite alphabet $\mathfrak{X}(x_1, \ldots, x_n)$ the following elementary events are defined $E_1 = x_1 \vee x_2 \vee \ldots \vee x_n$, $E_2 = x_1x_1 \vee x_1x_2 \vee \ldots \vee x_ix_j \vee \ldots \vee x_nx_n$, $E_{x_i} = \{E_1\} x_i$ and $E'_{x_i} = E_1 \vee \{E_1\} x_iE_1$, $(i = 1, 2, \ldots, n)$. It can be proved that those and only those events can be represented in finite automata, which are either among the elementary events enumerated or can be obtained from them by the application of a finite number of operations of disjunction, intersection, extension, contraction and superposition of elementary events $x_1, \ldots, x_n$, with the events obtained in this way. The last operation is called substitution and consists of the substitution (not necessarily uniquely) for the letters $x_1, \ldots, x_n$, of the same letters (or some of them) arranged in another order.

In one respect, this result is a direct consequence of the results stated above (theorems 18-21).

An interesting approach to the representation of events in finite automata can be made using the concept of the direct transfer matrix of a finite automaton, introduced in the work of Burks and Wang [3].

The matrix of direct transfer of a finite automaton A is a square matrix, the columns and rows of which are associated with different states of the automaton. The order of this matrix is thus equal to the number n of the states of A. The states of the automaton may be denoted by the sequence of numbers 1, 2, ..., n. The element α_{ij} of the matrix is the disjunction of all the words of length 1 (i.e, single letters), transforming the automaton from the state i to the state j, or the impossible event ϕ, if words with the requisite properties are missing. It is in the ith row and the jth column.

For the automaton shown in fig. 1 §1, assuming that the state a appears first and the state b second, the matrix of direct transfer C will be

$$C = \begin{bmatrix} \phi & x \vee y \\ \phi & x \vee y \end{bmatrix}.$$

Using the operations of multiplication and disjunction of the algebra of events, we can define the product D of two square matrices $A = [\alpha_{ij}]$ and $B = [\beta_{km}]$ of the same order n, the elements of which are events defined by the same alphabet used for A and B. The product D is defined as a square matrix of the nth order, the (i, m)th element of which is defined as the event $\gamma_{im} = \alpha_{i1}\beta_{1m} \vee \alpha_{i2}\beta_{2m} \vee \ldots \vee \alpha_{in}\beta_{nm}$.

Since the elements of the direct transfer matrix C of an automaton are events in the input alphabet of the automaton we can define powers of this matrix: $C^1 = C$, $C^2 = CC$, $C^3 = C^2C \ldots$, $C^k = C^{k-1}C$, ...

THEOREM 22. *For any integer k the ijth element of the matrix C^k is the event equal to the disjunction of all words of the input alphabet of A having length k which transfer A from the state i to the state j.*

The proof of this theorem follows directly by induction with respect to k using the relations (7) and (8) given above.

The disjunction of the matrices C_1 and C_2 of the same order is defined as the matrix of the same order, each element of which is the disjunction of the corresponding elements of C_1 and C_2. The unit matrix is defined to be the square matrix, the elements of the principal diagonal of which are the events consisting of the empty words, and the non-diagonal elements are the impossible event. It is natural to take the zero power C^0 of any transfer matrix to be the unit matrix of the same order.

The complete transfer matrix of a finite automaton A is defined as the infinite disjunction $C^0 \vee C \vee C^2 \vee \ldots \vee C^k \vee \ldots$ of all positive integer powers of the transfer matrix of A.

THEOREM 23. *The ijth element of the complete transfer matrix is the event which represents all those words of the input alphabet of A, which transfer the automaton from the state i to the state j.*

Naturally, theorems 22 and 23 also hold for infinite automaton, but their effective application (for the purpose of actually describing the representation of events by an automaton) is confined, as a rule, to finite automata. Indeed, one can talk only in a conventional sense of the effective applicability of theorem 23 even for finite automata, as the direct determination of the regular expressions for the elements of the complete transfer matrix, based on the definition of this matrix given above, is not always possible.

Elements of the complete transfer matrix are events representing a single state of the automaton. Events represented by sets of states must be written down as the disjunction of such elements.

In addition to this a question arises about the existence of events not represented by a single state and about the intrinsic nature of such events.

A means of answering this was given by Bodnarchuk. If there exists for a certain event S two words p and r, so that the word p, as well as the word pr belong to this event, then the word r is called a *bundle* of the event S (the bundle may or may not appear explicitly in the event). The word q is called a *cycle* of the event S, if for any arbitrary word l of the event S, the word lq also belongs to this event.

Bodnarchuk states the following theorem.

THEOREM 24. *An event, which contains a bundle which is not a cycle cannot be represented by a single state in any automaton.*

PROOF. Let the event S be represented by a single state a in an automaton A, which may be taken to be an initial automaton with initial state a_0 which is transformed by all the words of the event S into the state a. Let r be the arbitrary bundle of the event S. By the definition of a bundle a word $p \in S$ exists such that $pr \in S$. But $a_0p = a$ and $a_0pr = a$; thus $ar = a$. If now, q is any arbitrary word from S, then $a_0q = a$ and $a_0qr = ar = a$, in other words, $qr \in S$ and consequently the word r is a cycle. This proves the theorem.

Using theorem 24, it is easy to prove that the event $S = x \vee xy$, cannot be represented by a single state in any automaton. Indeed, the word y evidently represents the bundle belonging to this event, which at the same time is not a cycle, so long as the word xyy is not contained in S.

Finally we mention the problem of the definition of the number of non isomorphic finite automata with given input and output alphabets and with a given number of states. This has been solved by Murray [18] in the case of an automaton without output signals and a single letter input alphabet.

§ 4. Automata and semi-groups

In the present section, we determine the connections between the theory of automata and the theory of semi-groups. This connection has already been examined in part in previous sections, but only as a matter of terminology restricted to free semi-groups. An attempt to define a closer connection between the theory of semi-groups and the theory of automata is given in [11]. This is further developed below.

Let us consider an arbitrary automaton A with an input alphabet $\mathfrak{X}$. Each letter x of the alphabet $\mathfrak{X}$ (an input signal of automaton A) gives rise to a certain interchange (in general not one-one) in the states of A. This interchange associates the state $ax = \delta(a, x)$ with the arbitrary state a. ($\delta(a, x)$ is the transfer function of A). This substitution is called the substitution induced by the input signal x.

The semi-group generated by all these substitutions induced by the various input signals of the automaton A is called the semi-group associated with the automaton, the automaton is said to be an automaton belonging to the given semi-group. Semi-groups associated with automata are usually considered (if not stated otherwise) as abstract semi-groups, and not as semi-groups of substitutions.

The following theorem holds.

THEOREM 25. *The semi-groups of isomorphic automata are mutually isomorphic.*

Let A and B be two mutually isomorphic automata. Let Φ denote an isomorphism mapping of A onto B. For any arbitrary state a of A and any arbitrary letter x of its input alphabet $\mathfrak{X}$, $\varphi(a)$ and $\varphi(x)$ denote the corresponding state and the corresponding letter of the input alphabet $\mathfrak{X}_1$ of automaton B with respect to the isomorphism φ. If $p \doteq x_1x_2 \ldots x_k$ is any arbitrary word in the alphabet $\mathfrak{X}$, the word $\varphi(x_1)\varphi(x_2) \ldots \varphi(x_k)$ in the alphabet $\mathfrak{X}_1$ is denoted by $\varphi(p)$.

By the definition of isomorphism of automata, $\varphi(ax) = \varphi(a)\varphi(x)$. Hence by induction, with respect to the length of the word, $\varphi(ap) = \varphi(a)\varphi(p)$.

The input word p induces a certain substitution $a \to ap$ of the states of A. It is easily seen that this substitution coincides with the product of substitutions, induced by the input signals $x_1, \ldots, x_k$ (the letters of the word p). Any defining relation of the semi-groups of the automaton A can be written in the form $ap = aq$, where p and q are words in the alphabet $\mathfrak{X}$, while the state a runs through the set $\mathfrak{A}$ of all the states of automaton A. The relation $\varphi(a)\varphi(p) = \varphi(a)\varphi(q)$ in the automaton B corresponds to the previous relation in the automaton A. Using besides the mapping φ, also the inverse mapping φ^{-1} we establish a one-one correspondence between the sets of generating elements and defining relations of the semi-groups of automata A and B, which implies isomorphisms of these semi-groups. This proves theorem 25.

Let us now consider an arbitrary homomorphism φ of the automaton A to the automaton B. Relating each substitution, induced by an input signal x in A to the substitution, induced by the input signal $\varphi(x)$ in B and the products of substitions of the first kind to the products of corresponding substitutions of the second kind, we define a certain mapping of the semi-group of A on the semi-group of B. This mapping is called the mapping induced by the homomorphism φ.

The method used to prove theorem 25 can be used to prove the following theorem.

THEOREM 26. *The mapping ψ induced by the homomorphism φ of an automaton A to the automaton B, is the homomorphism of the semi-group G of A to the semi-group H of B. If the homomorphism φ maps the automaton A onto the automaton B, then the homomorphism ψ also generates a mapping of semi-group G onto the whole semi-group H. If homomorphism φ is an isomorphism, then ψ will also be an isomorphism.*

We can approach the concept of the semi-group of an automaton from another side, by studying certain special subdivisions of free semi-groups.

As usual, we mean by a *partition* of a set M a family of non-empty non-intersecting subsets $\{M_\alpha\}$ of M whose union is M. We shall say that the partition $\{N_\beta\}$ is *inscribed* in the partition $\{M_\alpha\}$ of the same set if each set N_β of the first partition is contained in some set M_β of the second partition; we also say, in this case, that $\{N_\beta\}$ is a *refinement* of $\{M_\alpha\}$.

By an ascending chain of partitions of a set M we shall mean a totally ordered set of partitions such that each partition of the set is a refinement of its successors. The union of an ascending chain of partitions R_j of a set M is the partition R which is such that two elements of M belong to the same set of R if and only if there is a set in some partition R_j to which they both belong.

We may define two operations for partitions of a set M, the operations of intersection and of union of partitions.

If $G = \{G_\alpha;\ \alpha \in A\}$ and $H = \{H_\beta;\ \beta \in B\}$ are two arbitrary partitions of the set M then the intersection of the partitions G and H is defined as the partition, composed of all non-empty intersections $G_\alpha \cap H_\beta$ $(\alpha \in A,\ \beta \in B)$ of the sub-sets of the first and second partitions.

To define the union of the partitions $G = \{G_\alpha;\ \alpha \in A\}$ and $H = \{H_\beta;\ \beta \in B\}$ of the set M, we introduce the concept of a GH-chain. A GH-chain connecting the elements p and q of the set M_0, is any finite sequence $p = m_0,\ m_1,\ m_2,\ \ldots,\ m_{k-1},\ m_k = q$ of elements of M, such that for any $i = 1,\ 2,\ \ldots,\ k$ the elements m_{i-1} and m_i are contained either in the same subset G_α of the partition G, or in the same subset H_β of partition H.

A subset N of a set M is called a GH-connected subset if any two of its elements can be connected by a GH-chain, and a maximal GH-connected subset if, in addition, it is not contained in any larger GH-connected subset of the set M.

Any element m of the set M is contained in a certain maximal GH-connected subset, namely the set of all the elements which can be connected by GH-chains with the element m. It follows directly from the definition of the GH-chain that the union of GH-connected subsets, which contain a common element form a GH-connected subset. Thus maximal GH-connected subsets do not intersect one another.

It can thus be shown that the aggregate of all maximal GH-connected subsets of the set M is a partition of this set. This partition is called the union of the partitions G and H. It follows directly that each of the partitions G and H is a sub-partition of their union.

We introduce the following

DEFINITION 16. The partition $K = \{K_\alpha,\ \alpha \in A\}$ of the free semi-group F in an alphabet $\mathfrak{X}$ is called *automatonic*, if for any letter x of the alphabet $\mathfrak{X}$ and any subset K_α of the partition K, the subset $K_\alpha x$ is contained in some subset of this partition. The partition K is called a *semi-group partition*, if it is defined by some relation of congruence in a semi-group F, in other words, if the product $K_\alpha K_\beta$ of any arbitrary two subsets of the partition K is contained in a third subset of this partition (depending on α and β).

It is easy to see that every semi-group partition of a free semi-group is an automatonic partition, but the converse is not generally true. By analogy with the given definition of an automatonic partition given above, which it is natural to call *right-automatonic*, we may also define left-automatonic partitions $\{K_\alpha,\ \alpha \in A\}$, for which we have the inclusion $xK_\alpha \subset K_\beta$. It is clear that each semi-group partition will not be only right- but also left-automatonic. However, in the following, we shall only use the right-automatonic partitions.

In the present section the term "free semi-group" is taken to mean both free semi-groups (without a unit element), and free semi-groups with a unit element (the empty-word). If any proposition applies to only one type of free semi-group, the fact is mentioned. If no special mention is made, a proposition applies to both types.

THEOREM 27. *The intersection and the union of the partitions G and H of the free semi-group F are automatonic partitions, if both the partitions G and H are automatonic, and are semi-group partitions if both partitions G and H are semi-group partitions.*

PROOF. The theorem is obvious in the case of the intersection of partitions. Consider the case of the union, which may be written $Q = \{Q_\gamma,\ \gamma \in P\}$. We assume that both partitions

$$G = \{G_\alpha;\ \alpha \in M\} \text{ and } H = \{H_\beta;\ \beta \in N\}$$

are automatonic. Let a and b be two arbitrary elements from the arbitrary set Q_γ of the partition Q, $a_0 = a,\ a_1,\ a_2,\ \dots,\ a_n = b$, the connecting GH-chain, and let x be an arbitrary letter of the alphabet of the semi-group F (i.e. an arbitrary generating element of this semi-group). Since the partitions G and H are automatonic, it follows at once that $a_0x,\ a_1x,\ \dots,\ a_nx$ are elements of a GH-chain, connecting the elements $a_0x = ax$ and $a_nx = bx$. Since ax and bx are arbitrary elements of the set $Q_\gamma x$, this set is GH-connected. It is therefore contained in a certain maximal GH-connected subset, i.e. in one of the sets of the partition Q. This proves that partition Q is automatonic.

Let us assume that G and H are semi-group partitions. Let a and b be an arbitrary pair of elements of the product $Q_{\gamma_1}Q_{\gamma_2}$ of two arbitrary sets of the partition Q. Then $a = cp$, $b = dp$, where $c,\ d \in Q_{\gamma_1}$, $p,\ q \in Q_{\gamma_2}$. Let us construct a GH-chain $c_0 = c,\ c_1,\ c_2,\ \dots,\ c_m = d$ and $p_0 = p,\ p_1,\ p_2,\ \dots,\ p_n = q$ joining the elements c and d and the elements p and q respectively. Then the chain $c_0p_0 = cp,\ c_1p,\ c_2p,\ \dots,\ c_mp = dp,\ dp_1,\ dp_2,\ \dots,\ dp_n = dq$ is a GH-chain, connecting the elements cp and dq.

In fact, the elements c_{i-1} and c_i for any arbitrary $i = 1,\ \dots,\ m$ are contained in the same set G_{α_i} or H_{β_i}. If the partitions G and H are defined by a congruence relation, then the set $G_{\alpha_i}p$ is contained in one of the sets of the partition G, while the set $H_{\beta_i}p$ is in one of the sets of partition H. Consequently, both elements $c_{i-1}p$ and c_ip are contained simultaneously in one of the two last sets. Using the same reasoning for any arbitrary pair of elements $dp_{j-1},\ dp_j$ $(j = 1,\ \dots,\ n)$, we prove that the chain constructed above is a GH-chain.

It is proved, similarly, that the set $Q_{\gamma_1}Q_{\gamma_2}$ is GH-connected and, consequently, is contained in one of the maximal GH-connected sets.

Since all such sets appear in the partition Q, the theorem is proved.

We say that an automatonic (or semi-group) partitiom K of a free-semi-group F is a maximal automatonic (or semi-group) partition in a given set of partitions R_α of the semi-group F, if the partition K is inscribed in all partitions R and is not a sub-partition of any larger automatonic (or semi-group) partition inscribed in all partitions R_α (the notion of maximality refers in this to the sets composing the partition and not to the partition itself).

THEOREM 28. *For any set of partitions of the free semi-group F there exists one and only one maximal automatonic (or semi-group) partition inscribed in all partitions of this set.*

PROOF. The partition of the semi-group F into separate elements (words) is a semi-group partition and consequently automatonic, which is inscribed in any partition of this semi-group belonging to the set M. It is also clear that the union of an arbitrary continued chain of automatonic (or semi-group) partitions is itself a semi-group partition. Therefore a maximal automatonic (or semi-group) partition inscribed in every set of M exists. We may denote it by G.

If, besides the partition G, there were a second maximal partition H, inscribed in all the partitions of M, then by theorem 27 their union Q would be an automatonic or semi-group partition according to whether partitions G and H are automatonic or semi-group partitions. It is easy to see that the partition Q is inscribed in any partition R of the set M. Indeed, any two arbitrary elements a and b, appearing in different subsets R_α and R_β of the partition R, cannot be connected by a GH-chain, since any arbitrary subset of the partitions G and H is either contained in or does not meet the sets R_α and R_β respectively. Therefore any arbitrary GH-connected subset, in particular, any subset of the partition Q is contained in one of the subsets of the partition R. Finally, we see that each of the partitions G and H is itself a sub-partition of the partition Q, and that the partition Q is strictly greater than both the partitions G and H if these two partitions do not coincide. As this is impossible the partitions G and H are identical.

The automatonic partitions of a free semi-group are a convenient method of studying so-called *connected* initial automata.

DEFINITION 17. A certain state a_2 of an automaton A is *generated* by the state a_1 of the same automaton, if a word p can be found in the input alphabet of A, transforming the automaton from the state a_1 into the state a_2. The initial automaton is said to be *connected*, if all its states are generated by the initial state.

In the present section we shall consider automata, defined only by sets of states and of input signals, connected to one another by the transfer function. We shall not consider output signals and the output function. We shall call such automata automata without output signals. An automaton without output signals can be considered as a Moore automaton, in which the location of a state is the state itself, and consequently the set of output signals coincides with the set of states. Such automata will be called *Medvedyev automata*.

The representation of an arbitrary connected initial automaton A with an input alphabet $\mathfrak{X}$ defines a certain automatonic partition K of the free semi-group F with unit element in the alphabet $\mathfrak{X}$, namely, the partition consisting of the set of events, represented by the separate states of A. Indeed, if F_α is an arbitrary set of the partition K (consisting of all the words in the alphabet $\mathfrak{X}$, transferring A from the initial state a_0 to a certain state a_α), and x is an arbitrary letter of the input alphabet $\mathfrak{X}$, then the set $F_\alpha x$ is contained in one of the sets of the partition K (namely, in the set of words, transferring the automaton from the state a_0 into the state $a_\alpha x$). The automatonic partition K, obtained in this way will be called the partition corresponding to the given automaton A.

Conversely, for each automatonic partition K of a free semi-group F (with a unit element) in the alphabet $\mathfrak{X}$, we can define uniquely, apart from an $\mathfrak{A}$-isomorphism, a connected initial automaton A (without output signals) for which K will be the partition corresponding to it. Indeed, we take for the states of the automaton A the sets F_α, which compose the partition K themselves. The transfer function of A can be uniquely defined in the following way: for any arbitrary letter x of the input alphabet and any arbitrary state F_α of the automaton A, the state $F_\alpha x$ is taken to be that the set of the partition K which contains the set $F_\alpha x$.

Taking as the initial state of the automaton the set F_0, containing the empty word e, we define A as an initial automaton without output signals. If p be any word of any arbitrary set F_α of the partition K, then in the automaton A this word transfers A from the state F_0 to the state F_α. In fact if $e \in F_0$, while $ep = p \in F_\alpha$, then by the definition of an automatonic partition $F_0 p \subset F_\alpha$. Similarly it may be proved that the automaton A is connected and the initial partition K serves as the partition corresponding to the automaton A.

Let B be any other connected initial automaton without output signals, to which the partition K corresponds. Setting up a correspondence between each state b of B, and the state a of A, representing the same event, it is clear that we have an $\mathfrak{A}$-isomorphism of the automaton B on the automaton A.

THEOREM 29. *There is a naturally defined one-one correspondence between the set of all connected initial automata (without output signals), no two of which are isomorphic with respect to states*, in any given input alphabet $\mathfrak{X}$ and the set of all automatonic partitions of the free semi-group with unit in the alphabet $\mathfrak{X}$.*

Using theorem 25, we shall identify connected initial automata without output signals with the partitions, corresponding to them, of free semi-groups.

A given event S in an alphabet $\mathfrak{X}$ determines a partition of the free semi-group F in the alphabet $\mathfrak{X}$ onto two sets S and $\bar{S}$. The problem of the synthesis of an initial automaton, representing a given set of events $\{S_\alpha;\ \alpha \in M\}$ by sets of its states, can be formulated in the language of partitions as the problem of the inscribing an automatonic partition in the set of partitions, defined by the events S_α $(\alpha \in M)$.

We can reduce this problem to that of inscribing an automatonic partition in a particular partition R, consisting of all non-empty intersections

* Editor's note: that is, the set of all equivalence classes of such automata, equivalence being defined by isomorphism.

of the form $\bigcap_{\alpha \in M} \tilde{S}_\alpha$, where $\tilde{S}_\alpha$ denotes either S_α, or $\overline{S}_\alpha$. The same may also be done (Letichevskii [36]), by constructing a sequence of sub-partitions from the partition $R = \{R_\beta;\ \beta \in N\}$. For this purpose each of the subsets R_β is decomposed into non-intersecting subsets R_{β_γ}, such that for each letter x of the input alphabet $\mathfrak{X}$ the product $R_{\beta_\gamma}x$ is contained in one of the sets $R_\beta(\beta \in N)$. The set R_{β_γ} in its turn, is likewise decomposed by the same rule. Proceeding in this way, we obtain the required automatonic partition.

This process would lead to an effective synthesis algorithm for finite systems of regular events, if there were a sufficiently simple means of finding regular expressions for the intersection and the complement of regular events.

Unfortunately no known methods of solving this problem exist at present, although it is essentially more simple than the method based on the synthesis of automata given in theorem 18.

We shall now discuss semi-group partitions.

Each automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ defines a certain semi-group partition of the free semi-group F in the alphabet $\mathfrak{X}$. In fact, each word p in F corresponds to a certain mapping $a \rightarrow ap(a \in A)$ of the set $\mathfrak{A}$ of the states of automaton A on itself. For each such mapping φ we denote by S_φ the set of all words in the alphabet $\mathfrak{X}$ which generate the mapping φ.

The system of all non-empty sets S_φ gives, obviously, a certain partition R of the semi-group F. Also it is easy to see that $S_\varphi S_\psi \subset S_{\varphi\psi}$. The partition R is therefore a semi-group partition. It may be called the semi-group partition defined by the automaton A. The partition R defines a certain congruence relation on the free semi-group F, which we shall call the congruence relation induced by the given automaton A. From the relation $S_\varphi S_\psi \subset S_{\varphi\psi}$ and the method by which the partition R is defined we have the following theorem.

THEOREM 30. *The factor semi-group for the congruence relation induced by any given automaton A is isomorphic to the semi-group of that automaton.*

Consider an arbitrary connected initial automaton A with an input alphabet $\mathfrak{X}$ and the corresponding partition R of the free semi-group F in the alphabet $\mathfrak{X}$, i.e. in other words, the set of events $\{R_\alpha\}$ represented by the separate states a_α(a_α runs through the set $\mathfrak{A}$ of all the states of A). Let $P = \{P_\beta\}$ be the semi-group partition, defined by A, and $Q = \{Q_\gamma\}$ an arbitrary semi-group partition, inscribed in the automatonic partition R.

We shall show that the partition Q is a sub-partition of the partition P. Let Q_γ be an arbitrary subset of partition Q. We select an arbitrary word q, contained in Q, and consider the subset P_β of partition P containing the word q. For any subset R_α of the partition R we denote by $R_{\alpha(q)}$ that one of the subsets of this partition into which the subset R_α is transferred under the influence of the word q, $R_\alpha q \subset R_{\alpha(q)}$. From the definition of a partition P it follows that the subset P_β consists of all the words p such that $R_\alpha p \subset R_{\alpha(q)}$ for any arbitrary subset R_α of the partition R.

Since the partition Q can be inscribed in the partition R, each of the sets R_α will be the union of a certain set of subsets of the partition Q. Any such arbitrary subset $Q_{\gamma(\alpha)}$ is transformed by the word p into the set $R_{\alpha(q)}$: $Q_{\gamma(\alpha)}p \subset R_{\alpha(q)}$. If the partition Q is a semi-group partition the product $Q_{\gamma(\alpha)}Q_\gamma$ is contained in one of the subsets of Q, and therefore in one of the subsets of R. Since $p \in Q_\gamma$ and $Q_{\gamma(\alpha)}p \subset R_{\alpha(q)}$, then $Q_{\gamma(\alpha)}Q_\gamma \subset R_{\alpha(q)}$. This inclusion applies for any arbitrary subset $Q_{\gamma(\alpha)}$, contained in R_α. Consequently $R_\alpha Q_\gamma \subset R_{\alpha(q)}$.

Remembering that P_β consists of all words p, for which $R_\alpha p \subset R_{\alpha(q)}$, we see that $Q_\gamma \subset P_\beta$. Since the choice of the subset Q_γ from Q is arbitrary, it follows that the partition Q can be inscribed in the partition P.

THEOREM 31. *The semi-group partition, defined by any arbitrary connected initial automaton A, coincides with the maximal semi-group partition inscribed in the automatonic partition defined by the system of all the events, represented by the separate states of the automaton A.*

Consider the arbitrary system M of events $\{S_\alpha\}$ in an alphabet $\mathfrak{X}$ and the corresponding system K of partitions $\{S_\alpha, \overline{S}_\alpha\}$ of the free semi-group F with a unit element in the alphabet $\mathfrak{X}$. The unique (because of theorem 28) maximal automatonic partition inscribed in all the partitions of the system K will be called the automatonic partition defined by the system of events M. Similarly the unique maximal semi-group partition P, which can be inscribed in all partitions of K will be called the semi-group partition defined by the system of events M.

The semi-group partition P is defined by a certain congruence relation on the free semi-group F, which will be called the congruence relation defined by the given system of events M. Finally the factor-semi-group associated with this congruence relation (considered as an abstract semi-group) will be called the semi-group, defined by the given system of events M, while the system M itself will be called the system of events, belonging to that semi-group.

Let M be an arbitrary system of events in an alphabet $\mathfrak{X}$. The automatonic partition $R = \{R_\beta\}$ of the free semi-group F with a unit element defined by this system, defines uniquely (by theorem 29) a certain connected initial automaton A, the states of which one may take to be sets R_α of the partition R. The transfers produced by the letters $x \in \mathfrak{X}$ are then defined in such a way, that the state $R_\alpha x$ is chosen to be that set of the partition R which contains the set $R_\alpha{}'x$. The initial state is chosen to be the one that contains the empty word e.

It is easy to see that in a connected initial automaton A without output signals, constructed in this way, all events of the system M can be represented by certain sets of states. This automaton is called the automaton defined by the initial system of events M.

Let B be any other connected initial automaton without output signals in the alphabet $\mathfrak{X}$, representing all events of the system M by sets of its states. As for the automaton A, we can identify the states of automaton B with the events Q_γ, represented by its separate states. The set of all subsets Q_γ form an automatonic partition Q, inscribed in all partitions $(S_\alpha, \overline{S}_\alpha)$, defined by the events S_α of the system M. Since R is a maximal partition we see that Q is inscribed in the partition R.

It is easy to see that, on assigning to each set Q_γ the set R_α which contains it, we get a homomorphism with respect to states ($\mathfrak{A}$ -homomorphism) of B on A.

THEOREM 32. *Any connected initial automaton (without output signals), representing the events of a given system of events by sets of its states, is mapped in an $\mathfrak{A}$-homomorphism on the automaton defined by this system of events.*

Theorem 32 shows that to construct an automaton defined by a given system of events M, it is sufficient to construct any connected initial automaton A without output signals, representing all events of the system M by sets of its states and then, considering A as a Medvedyev i.e. as a special case of a Moore automaton, in which these states are their own locations, to minimize this automaton by using theorem 6.

This gives a practical method of constructing (at least for finite systems of regular events) automata and semi-groups defined by a given system of events. For this purpose it is sufficient to apply the methods of synthesis and of the minimization of automata, developed in the previous section.

Theorem 32 proves that it will always be possible to construct such an automaton by this method. The semi-group defined by this system of events may be found by the following theorem.

THEOREM 33. *The semi-group, defined by any system of events is isomorphic with the semi-group of the automaton, defined by this system.*

This follows directly from theorems 30 and 31, since the maximal semi-group partition inscribed in the automatonic partition of any arbitrary system of events coincides with the semi-group partition defined by this system.

These results facilitate the study of systems of events, belonging to special forms of semi-groups (finite, commutative, nilpotent etc.)

The results of the preceding section have the following as consequence.

THEOREM 34. *All finite systems of regular events and only such are systems which belong to finite semi-groups.*

DEFINITION 18. An event is called *commutative*, if for any arbitrary word p in it, the event contains all those words derived from p by arbitrary permutations of letters forming the word.

THEOREM 35. *Arbitrary systems of commutative events and only such systems can be represented by systems of events belonging to commutative semi-groups.*

PROOF. Let M be an arbitrary system of commutative events $\{S_\alpha\}$ in the alphabet $\mathfrak{X}$.

The partition R of the free semi-group F with a unit element in the alphabet $\mathfrak{X}$ into the sets obtained by adding to any arbitrary given word all permutations of its letters, is clearly a semi-group partition inscribed in all the semi-groups $\{S_\alpha, \overline{S_\alpha}\}$. The congruence relation corresponding to this partition is $xy = yx$, and defines the factor semi-group G, a free commutative semi-group.

It is clear that the partition R is inscribed in the semi-group partition P, defined by the system of events M. The congruence relation corresponding to the partition P defines the factor semi-group H, isomorphic by the theorem of homomorphism to a certain factor-semi-group of the semi-group G. Since this last semi-group is commutative, the semi-group H is commutative, which proves the theorem in one direction.

Conversely, if a system of events M belongs to a commutative semi-group, and if this semi-group is given in the form of a factor semi-group of a free semi-group defined by a certain congruence relation, the equivalence classes defined by this relation represent commutative events. Since the union of any commutative events is commutative, all events of the set M will be commutative. This proves theorem 35.

Theorems 34 and 35 give a sufficient account of the essence of the problems arising in the study of events and semi-groups by the methods described. The most interesting problem is the construction of equivalence classes corresponding to any arbitrary congruence relation for a free semi-group, which define the factor-semi-groups, isomorphic with a given abstract semi-group G.

It is clear, for example, that the presence of an equivalence class which is not a regular event would be sufficient to prove that the semi-group G is infinite. It is easy to see that similar considerations play an essential part in the solution of problems such as the well-known greater problem of Burnside.

The results of the present section make possible the extension of the language of regular notation for events and a sufficiently effective description of a large class of events, not representable in finite automata. This extension depends on using, besides regular expressions, both identical and general defining relations in a free semi-group.

DEFINITION 19. Let S be an arbitrary event in an alphabet $\mathfrak{X}$, and M the arbitrary system of defining relations in the free semi-group F in the alphabet $\mathfrak{X}$. The *closure* of the event S with respect to the system of relations M is the event consisting of all words of the event S and all words, equivalent to them by virtue of the defining relations of the system M. The *commutative closure* of an event is its closure with respect to the single identical defining relation $xy = yx$.

The result of commutative closure is that any event is transformed into the commutative event completed by the words derived from the words of the original event by all possible permutations of their letters.

It is easy to see that the commutative closure of a regular event is not necessarily regular. Indeed, applying the operation of commutative closure to the regular event $R = \{xy\}$, we obtain the event S, consisting of all words in the alphabet (x, y), which contain an equal number of letters x and y. By theorem 14, S cannot be represented in a finite automaton and consequently, is not regular (theorem 16).

In this way, the operation of commutative closure already leads outside the limits of the class of regular events. By using the operation of the closure of events defined by regular expressions, with finite systems of defining relations we get a language suitable for describing a class of events including in itself the class of regular events as a proper sub-class.

It may be noted that Sorkin [38] has considered the problem of specifying automata (without output signals) in terms of generating elements and defining relations. Thus for an arbitrary automaton $R(\mathfrak{A}, \mathfrak{X}, \delta)$ without output signals, the set of states $\mathfrak{B} \subset \mathfrak{A}$ is selected such that any state a of automaton A is generated by at least one state $b \in \mathfrak{B}$. In other words, an input word p exists such that $bp = a$.

A set $\mathfrak{B}$ is naturally called a set of generating states of A. If we exclude the case of free automata, examined earlier (definition 9), then in the automaton A we can find at least one state a, which can be obtained in various ways from the generating states, for example $a = b_1p_1$ and $a = b_2p_2$. This circumstance naturally arises introducing the defining relation $b_1p_1 = b_2p_2$.

From the defining relation obtained in this way, we can derive further defining relations by right multiplication of both sides of this relation by any arbitrary input word $q : b_1p_1q = b_2p_2q$. Using this circumstance we can for a specified automaton A write down not all the defining relations, but only a set R, such that any defining relation of A is either in R or can be formed as a derived relation of some defining relation in R. Every such set R is called a system of defining relations of the automaton A (with respect to the given system of generating states $\mathfrak{B}$).

It is easy to see that every automaton without output signals $A(\mathfrak{A}, \mathfrak{X}, \delta)$ can be given by a system of generating states $\mathfrak{A}$ and a system of defining relations $ax = \delta(a, x)$, where a runs through the set of all the states of the automaton and x through its input alphabet $\mathfrak{X}$. Defining relations of the form $ax = b$, where x is the letter of the input alphabet will be called canonical.

If the automaton A is specified by an arbitrary system of generators $\mathfrak{B}$ and by the system of defining relations R, it is not difficult, following Sorkin, to define this automaton using only canonical defining relations, introducing new generating states. Thus given the relation $ap = bq$, where $p = p_1x_1$ we can introduce the new states $a_1 = ap$, and $b_1 = bq$ and transform this relation into a canonical relation $a_1x = b_1$. In this way Sorkin solves, for automata specified by a finite systems of defining relations, the problem of the identity of words and the problem of isomorphism. The idea of the construction of algorithms for solving these problems, consists in transforming the system of defining relations to a canonical form, in the rejection of sets of states belonging to free sub-automata and in the study of the finite partial automata left after these operations. We do not go into details.

§5. The composition of automata

In the systematic theory of automata a large part is played by various methods known as the composition of automata, which allow the construction of more complicated automata from simpler ones. We shall first consider the *direct sum* of automata.

DEFINITION 20. *The direct sum* of the set of automata $A_\alpha(\mathfrak{A}_a, \mathfrak{X}, \mathfrak{Y}, \delta_\alpha, \lambda_\alpha)$ $(\alpha \in M)$ with common input and output alphabets is the automaton $B(\mathfrak{B}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ whose set of states coincides with the union of the sets of states of all automata $A_\alpha (\alpha \in M)$, while the values of the transfer function $\delta(a, x)$ and output function $\lambda(a, x)$ coincide with the values of $\delta_\alpha(a, x)$ and $\lambda_\alpha(a, x)$ in that automaton A_α, which has the state a in its set of states.

The above definition can be used in a natural way in the decomposition of a given automaton into the direct sum of its $\mathfrak{A}$-sub-automata. Such a decomposition is important in the study of an important class called *semi-simple automata.*

We now introduce the conception of the *kernel* of an automaton for the definition of a semi-simple automaton. Let $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ be an arbitrary automaton. Consider the set $\mathfrak{A}_1$ of all those states of A which are values of the transfer function of A. Denoting by $\mathfrak{A}\mathfrak{X}$ the set of all products ax $(a \in \mathfrak{A},\ x \in \mathfrak{X})$, we obtain $\mathfrak{A}\mathfrak{X} = \mathfrak{A}_1$. Construct the $\mathfrak{A}$-sub-automaton $A_1(\mathfrak{A}_1, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$ whose set of states is set $\mathfrak{A}_1$, while the transfer and output functions coincide with those of A to the set $\mathfrak{A}_1 \times \mathfrak{X}$. The automaton A_1 is called the *kernel*, or more correctly the *first kernel* of A.

DEFINITION 21. An automaton which coincides with its kernel is called a semi-simple automaton. An automaton is called simple, if for any pair (a, x) of its states (not necessarily different), there exists a non-empty input word, transferring the automaton from the state a to the state b.

Simple automata under the name strongly connected automata were first defined by Moore [20]. The name strongly-connected for these automata was suggested by their relation to the connected automata studied in the previous section. In a connected (initial) automaton all states can be generated by one particular state, but in a strongly connected automaton any arbitrary state can generate all the states of the automaton.

Strongly connected automata can also be characterized as not possessing proper $\mathfrak{A}$ sub-automata.

THEOREM 36. *Any arbitrary simple (strongly connected) automaton and the direct sum of any arbitrary set of simple automata are semi-simple automata.*

This follows directly from definition 21.

The reversible automaton considered by Huzino [30] is a special case of a semi-simple automaton. The automaton A is called reversible, if for any arbitrary state a and any arbitrary input word p, we can find an input word q transferring the automaton from the state ap to the state a. The word q is assumed to depend on the word p and on the state a. This differs from Huzino's definition who assumed that the word q was independent of the state a. Every automaton reversible in Huzino's sense will also be reversible in our sense. The converse is not true: it is easy to construct an automaton with two states reversible in our sense but not in the Huzino sense.

An automaton will be reversible in the Huzino sense when the semigroup of this automaton is a group. These automata may be called group automata, leaving the term reversible for the more general class of automata introduced above. The following theorem generalizes Huzino's result.

THEOREM 37. *Every reversible automaton can be decomposed into a direct sum of simple strongly connected automata.*

PROOF. Let $A = A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ be an arbitrary reversible automaton. Let us consider any state a_1 and the set $\mathfrak{A}_1$ of all the states which are generated by the state a_1. Denoting by δ_1 and λ_1 the functions δ and λ restricted to the set $\mathfrak{A}_1 \times \mathfrak{X}$, we obtain an $\mathfrak{A}$-sub-automaton

$A_1 = A_1(\mathfrak{A}_1, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$ belonging to A. This sub-automaton is strongly connected. Any arbitrary pair of its states can be represented in the form a_1p and a_1q, where p and q are certain input words of A. By the definition of reversibility there exists an input word r, such that $a_1pr = a$. The word rq thus transfers the automaton from the state a_1p to the state a_1q. If at least one of the states a_1p and a_1q is different from the state a_1, the word rq will be a non-empty word. If, however, $a_1p = a_1q = a_1$, then for any arbitrary word $x \in \mathfrak{X}$ either $a_1x = a_1$, or $a_1xl = a_1$ for a certain non-empty word l (if $a_1x \neq a_1$). Therefore in all cases the state a_1p is taken into the state a_1q by a certain non-empty word. Consequently the automaton A_1 is strongly connected.

If $\mathfrak{A}_1 = \mathfrak{A}$, then $A = A_1$, and the theorem is proved. If however $\mathfrak{A}_1 \neq \mathfrak{A}$, then we select an arbitrary state a_2 not contained in the set $\mathfrak{A}_1$, and we construct a sub-automaton $A_2(\mathfrak{A}_2, \mathfrak{X}, \mathfrak{Y}, \delta_2, \lambda_2)$, the set of whose states are the states generated in the automaton A by the state a_2. As above, we can show that the automaton A_2 is strongly connected. If the sets $\mathfrak{A}_1$ and $\mathfrak{A}_2$ intersect then any arbitrary element b from their intersection would allow a representation in the form $b = a_1p_1$ and $b = a_2p_2$, whence, by the property of reversibility, it is easy to deduce that $a_2 = a_1p_1r_1$ for a certain input word r_1. This would mean that $a_2 \in \mathfrak{A}_1$, which contradicts the choice of the state a_2. Consequently the sets $\mathfrak{A}_1$ and $\mathfrak{A}_2$ can not intersect.

Continuing in this way, we may construct a system of strongly connected sub-automata A_α, the sets of states of which are non-intersecting. The sum of all these sets is the whole set $\mathfrak{A}$. By definition A is decomposed into the direct sum of the automata A_α. This proves the theorem.

It is not difficult to show by an example that the generalization of theorem 37 for an arbitrary semi-simple automaton is impossible.

Using the concept of the kernel introduced above, we can construct a theory of automata by analogy with the theory of rings. Thus for any arbitrary automaton A, we can construct a decreasing series of its kernels A_α, where α runs through a certain set of ordinal numbers, beginning with $\alpha = 0$. The first member of this series, A_0, is A itself, while for any arbitrary α, $A_{\alpha+1}$ is defined as the kernel of A_α. For a limit number β the sub-automaton A_β is defined as the one whose set of states coincides with the intersection of the sets of states of all the sub-automata A_α, $\alpha < \beta$. The series finishes with the first number γ, for which $A_{\gamma+1} = A_\gamma$.

The automaton A_γ is called the *lowest kernel* of A and will be semi-simple. In a sense it corresponds to the radical in the theory of rings, although in the theory of rings the radical is not semi-simple, but a factor ring.

We now turn to consider another form of composition of automata, which may be called the *product of automata*. For simplicity we shall only consider the product of a finite number of automata. The results could however be generalized.

DEFINITION 22. The direct product of a number of automata $A_i = A_i(\mathfrak{A}_i, \mathfrak{X}_i, \mathfrak{Y}_i, \delta_i, \lambda_i)$ $(i = 1, 2, \ldots, k)$ is the automaton $A = A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$ whose sets of states and of input and output signals are the products of those of the corresponding sets for the automata A_i: $\mathfrak{A} = \mathfrak{A}_1 \times \mathfrak{A}_2 \times \ldots \times \mathfrak{A}_k$, $\mathfrak{X} = \mathfrak{X}_1 \times \mathfrak{X}_2 \times \ldots \times \mathfrak{X}_k$, $\mathfrak{Y} = \mathfrak{Y}_1 \times \mathfrak{Y}_2 \times \ldots \times \mathfrak{Y}_k$.

The transfer and output functions $a' = \delta(a, x)$ and $y = \lambda(a, x)$ of A are given by the relations $a'_i = \delta_i(a_i, x_i)$, $y_i = \lambda_i(a_i, x_i)$ $(i = 1, \ldots, k)$. Here $a' = (a'_1, a'_2, \ldots, a'_k)$, $a = (a_1, a_2, \ldots, a_k)$, $x = (x_1, x_2, \ldots, x_k)$, and $y = (y_1, y_2, \ldots, y_k)$.

In practical applications the direct product corresponds to the simultaneous consideration of a set of automata without any connections between them. Far more frequent in practice is the combination of automata by a so-called logical or combinatorial scheme. This can be included in the abstract conception of the general term product.

DEFINITION 23. The product of automata $A_i = A_i(\mathfrak{A}_i, \mathfrak{X}_i, \mathfrak{Y}_i, \delta_i, \lambda_i)$ $(i = 1, \ldots, k)$ is the automaton $A(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta, \lambda)$, defined by two given sets $\mathfrak{X}$ and $\mathfrak{Y}$ and two mappings: the mapping φ of the set $\mathfrak{A}_1 \times \mathfrak{A}_2 \times \ldots \times \mathfrak{A}_k \times \mathfrak{X}$ to the set $\mathfrak{X}_1 \times \mathfrak{X}_2 \times \ldots \times \mathfrak{X}_k$ and the mapping ψ of the set $\mathfrak{A}_1 \times \mathfrak{A}_2 \times \ldots \times \mathfrak{A}_k \times \mathfrak{X}$ to the set $\mathfrak{Y}$. The set of states of A coincides with the product $\mathfrak{A}_1 \times \mathfrak{A}_2 \times \ldots \times \mathfrak{A}_k$ of the sets of states of the automata $A_i (i = 1, \ldots, k)$. If $a = (a_1, \ldots, a_k)$ is an arbitrary state of A, while $x \in \mathfrak{X}$ is an arbitrary corresponding input signal of A, then the values of the transfer and output functions of A for the pair (a, x) are defined by the relations $\delta(a, x) = (a_1x_1, \ldots, a_kx_k)$ and $\lambda(a, x) = y$, where $(x_1, x_2, \ldots, x_k)$ is the image of the element $(a_1, a_2, \ldots, a_k, x)$ in the mapping φ, and y is the image of the same element in the mapping ψ. If all the automata A_i are initial, then A is also an initial automaton and the state $(a_1^0, \ldots, a_k^0)$ is selected as its initial state, where a_i^0 is the initial state of A_i $(i = 1, \ldots, k)$.

The mapping φ in definition 23 is the abstract expression for the scheme of inverse connections and the mapping ψ is the abstract expression of the output scheme constructed in the practical realization of the composition of automata.

In practical applications of the theory of automata, the *conditions of completeness* of systems of finite automata are very important. The problem of completeness of systems can be formulated for two important cases, which we may call the cases of isomorphic and homomorphic representation of automata.

DEFINITION 24. The system of finite automata M is called *isomorphically complete*, if for any arbitrarily specified finite automaton A there exists a finite product of automata, isomorphic to certain automata of the system M, which contains an $\mathfrak{A}$-subautomaton, isomorphic to the automaton A. The system M is called *homomorphically complete*, if for any arbitrary finite automaton A, there exists a finite product of automata, isomorphic to a certain automaton from the system M, which contains an $\mathfrak{A}$ -subautomaton with an $\mathfrak{A}$-homomorphic mapping on A.

The system of events Q is said to be representable in the automaton A, if it is possible to choose an initial state of A, in which all events of the system Q can be represented by certain sets of its states. It is easy to see that a system of events representable in A, will also be representable in any arbitrary automaton B, which can be mapped $\mathfrak{A}$-homomorphically on A. To establish this fact it is sufficient to choose as initial state of B a state which is mapped in a certain homomorphism φ: B on A to the initial state of A, and to choose as the representative set of states in B, the complete set of inverse images in the homomorphism φ

of the representing sets of states of the automaton A.

The following result is a modification of a theorem given by Letichevskii (theorem 2 [37]).

THEOREM 38. *A system M of finite automata is homomorphically complete, if and only if any arbitrary system of regular events is representable in finite products of automata isomorphic to the automata of the system M.*

The necessary and sufficient conditions of the homomorphic completeness of systems of finite automata were given by Letichevskii [37]. The system must contain at least one automaton A, in which there is a state a_1 and input signals x_1 and x_2, such that the states $a_1 = ax_1$, and $a_2 = ax_2$ are different and that input words p_1 and p_2 exist transferring these states into the state a. We shall not establish Letichevskii's result, but prove the following theorem which is close to it and easier to prove.

THEOREM 39. *In order that a system M of finite automata should be isomorphically complete, it is necessary and sufficient that it should contain at least one automaton A, which has two different states a and b and input signals x_1, x_2, x_3, x_4 (not necessarily different) such that $ax_1 = a$, $ax_2 = b$, $bx_3 = a$, $bx_4 = b$.*

To prove the necessity we construct a subautomation B from the finite product of automata $A_1, \ldots, A_k$ of the system M. B has the two states a and b with two input signals x and y; the transfer function is defined by the relations $ax = b$, $bx = a$, $ay = a$, $by = b$. By definition 23 the states a and b are finite sequences of states a_i and b_i of the A_i: $a = (a_1, a_2, \ldots, a_k)$, $b = (b_1, b_2, \ldots, b_k)$. Again by the same definition, $ax = b$, $bx = a$, $ay = a$, $by = b$ show that there exist four sequences of input signals of the A_i: $(x_1, x_2, \ldots, x_k)$, $(x'_1, x'_2, \ldots, X'_k)$, $(y_1, y_2, \ldots, y_k)$, $(y'_1, y'_2, \ldots, y'_k)$ such that for any arbitrary $i = 1, \ldots, k$, we have the relations $a_i x_i = b_i$, $b_i x_i = a_i$, $a_i y_i = a_i$, $b_i y'_i = b_i$. Since $a \neq b$ there will be at least one value of i for which $a_i \neq b_i$. But then the states a_i and b_i of A_i belonging to the system M satisfy the condition of the theorem. This proves the necessity of the condition.

To prove the sufficiency, consider the arbitrary finite automaton A with n states and k copies of the automata $A_1, \ldots, A_k$ each of which has two different states a_i and b_i and four input signals x_i, y_i, z_i, u_i such that

$$a_i x_i = a_i, \quad a_i y_i = b_i, \quad b_i z_i = a_i, \quad b_i u_i = b_i \qquad (i = 1, \ldots, k). \tag{12}$$

If $2^k \geqslant n$, we can identify each state a in A with a finite sequence of states a_i or b_i of the automata A_i $(i = 1, \ldots, k)$, so that different sequences would correspond to different states. We call these sequences codes of states. Since the relations (12) permit any transfers between the states a_i and b_i, the mapping φ in definition 23 can be selected so as to realize any arbitrary transfer function in the set of codes of the automaton A. Constructing the mapping φ in the proper way, we can realize any output function on this set.

The product of $A_1, \ldots, A_k$ defined by the mappings selected will obviously contain a subautomaton $\mathfrak{A}$-isomorphic to the automaton A. The set of codes of states serves as the set of states of this subautomaton. This proves the theorem.

As well as the general concept of the product of automata it is desirable to consider several special cases. We shall for simplicity limit the discussion to automata without output signals and possessing common input alphabets.

If $A_i(\mathfrak{A}_i, \mathfrak{X}, \delta_i)$ $(i = 1, \ldots, k)$ is any arbitrary set of such automata, we may consider the automaton $A(\mathfrak{A}, \mathfrak{X}, \delta)$ without output signals, the set of states of which coincides with the set $\mathfrak{A}_1 \times \mathfrak{A}_2 \times \ldots \times \mathfrak{A}_k$, while the transfer function is given by the relation $(a_1, a_2, \ldots, a_k)x = (a_1x, a_2x, \ldots, a_kx)$. The automaton A constructed in this way is called the $\mathfrak{A}$*-direct product* of $A_1, \ldots, A_k$. If all the automata A_i are initial, then A is also an initial automaton; and the initial state is chosen to be the sequence of the initial states of the automata $A_i (i = 1, \ldots, k)$.

THEOREM 40. *In the* $\mathfrak{A}$*-direct product of initial automata* $A_1, \ldots, A_k$ *without output signals, we may represent (by sets of states) all events, which can be obtained from the events represented in the automata* $A_1, \ldots, A_k$, *by the operations of union, intersection and complementation.*

As well as the direct product we may define the *semi-direct product* and the *crossed product* of automata. Consider two automata $A(\mathfrak{A}, \mathfrak{X}, \delta)$ and $B(\mathfrak{B}, \mathfrak{X}, \delta_1)$ without output signals, having a common input alphabet $\mathfrak{X}$. The *semi-direct product* of A and B is the automaton $C(\mathfrak{C}, \mathfrak{X}, \delta_2)$ whose set of states consists of the set of all ordered pairs $c = (a, b)$ of states of A and B $(a \in \mathfrak{A}, b \in \mathfrak{B})$, while the input alphabet coincides with the input alphabets of A and B. The transfer function of the automaton C is defined by the relation*

$$(a, b)\,x = (a\varphi(b, x),\ cx), \tag{13}$$

where x is an arbitrary element of $\mathfrak{X}$, and $\varphi(b, x)$ is a function on the set $\mathfrak{B} \times \mathfrak{X}$ with values in the set $\mathfrak{X}$. If (13) is replaced by

$$(a, b)\,x = (a\varphi(b, x), b\psi(a, x)), \tag{14}$$

the product is called the crossed product of A and B. Here $\varphi(b, x)$ and $\psi(a, x)$ are functions on the sets $\mathfrak{B} \times \mathfrak{X}$ and $\mathfrak{A} \times \mathfrak{X}$ with values in the set $\mathfrak{X}$.

It is easy to see that $\varphi(b, x)$ and $\psi(a, x)$ can be considered as transfer functions of certain automata the set of states of which belong to the set $\mathfrak{X}$, and whose input alphabets coincide respectively with the sets $\mathfrak{B}$ and $\mathfrak{A}$. In this way the crossed product of A and B defined by specifying two automata dual to the automata A and B in the sense that the input alphabet of A and B plays the role of sets of states of these automata and the states of A and B are their input alphabets. Both the semi-direct and the crossed product occur in practical methods of the composition of real automata. It would be interesting to study events representable in such products of finite automata.

* Editors note: This equation does not seem meaningful as it stands. Possibly the righthand side should be $(a\varphi(b, x), bx)$ (cf. def 25 and following paragraph).

Huzino [32] considers a product of automata defined differently from those already considered here. This product is defined for two automata A and B in which both the sets of states and the input alphabets coincide. The result is an automaton C with the same set of states $\mathfrak{A}$ and the same input alphabet $\mathfrak{X}$ as A and B. If $\delta_1(a, x)$ and $\delta_2(a, x)$ are the transfer functions of A and B, then the transfer function of C is defined by $\delta(a, x) = \delta_2(\delta_1(a, x), x)$. The question of completeness can be investigated in the case of this product along the lines discussed earlier.

We may observe that the product of automata in the Huzino sense does not appear to be an abstract expression of any method of composition of real automata used in practice. Other special operations on automata have been considered by Huzino [31] and [33].

The so-called *superposition of automata* is particularly important among the methods of composition of automata in practical use. From a practical point of view the superposition of automata means that the output signals of one automaton are used as the input signals of another. The abstract form of this method is as follows:

DEFINITION 25. Let there be given two automata $A_1(\mathfrak{A}, \mathfrak{X}, \mathfrak{Y}, \delta_1, \lambda_1)$ and $A_2(\mathfrak{A}_2, \mathfrak{Y}, \mathfrak{Z}, \delta_2, \lambda_2)$ such that the input alphabet of the second coincides with the output alphabet of the first. By the *superposition* $A_2(A_1)$ of A_1 and A_2 we mean an automaton $B(\mathfrak{B}, \mathfrak{X}, \mathfrak{Z}, \delta, \lambda)$ whose set of states coincides with the product $\mathfrak{A}_2 \times \mathfrak{A}_1$ of the sets if states of A_1 and A_2. For any arbitrary state $b = (a_2, a_1)$ of B and any arbitrary input signal x, the values of the transfer and output functions are given by $\delta(b, x) = (\delta_2(a_2, \lambda_1(a_1, x)), \delta_1(a_1, x))$, $\lambda(b, x) = \lambda_2(a_2, \lambda_1(a_2, x))$.

It is easy to see that the superposition of automata is a particular case of a product in the sense of definition 23. Moreover in the case of automata without output signals superposition is a particular case of the semi-direct product. This again demonstrates the importance of studying the semi-direct product.

The concept of superposition has a natural generalization. It can happen that the input alphabet $\mathfrak{X}$ of A coincides with the product $\mathfrak{Y}_1 \times \mathfrak{Y}_2 \times \cdots \times \mathfrak{Y}_k$ of the output alphabets of the automata $A_1, A_2, \ldots, A_k$. In this case it is not difficult, by analogy with definition 25, to define the superposition of the automata $A_1, A_2, \ldots, A_k$ and the automaton A. By using repeatedly such a generalized superposition we can construct *nets of automata.*

In constructing such nets one usually begins with a completely defined selection of so-called elementary automata and goes on to construct more complicated automata in the form of nets of elementary automata. The general theory of nets along with the usual generalized superposition allows the possibility of closed chains of inverse connections, which reduce to the following: some of the output signals of the net are used as input signals in the elementary automata which take part in the formation of these output signals.

The theory of nets of elementary automata is a part of the general theory of automata which may be called the structural theory of automata. This theory has been developed chiefly on the applied side in many hundreds of articles. The majority of these consider so-called *combination nets,* composed of automata each of which has only one state (this being true also for the entire net).

Recently chains of automata and above all so-called nerve nets have been studied by abstract algebraical methods. A thorough discussion was given by Skornyakov [26], [27] and Kleene [14] and [15]. We shall not pursue this matter further as it touches on the structure theory rather than on the abstract theory of automata.

§6. Experiments with automata

In §1 the state-word $a_1a_2 \ldots a_k$ and the output word $q = y_1y_2 \ldots y_k$ corresponding to an input word $p = x_1x_2 \ldots x_k$ in an arbitrary state a of an automaton A were defined. The pair of words (p, q) obtained in this manner may be called an experiment of length k with the automaton A in the state a. The word p is called the *input word* and the word q the *output word* of the experiment and the word q is the *output word* or the *result* of the experiment.

For a Moore automaton a different definition may be given of the term experiment [20]. Like an ordinary experiment, a Moore experiment with a Moore automaton in any one of its states a defines a pair (p, q), consisting of an input word $p = x_1x_2 \ldots x_k$ and a corresponding output word q. The difference from the ordinary experiment is that in the Moore experiment the output word does not coincide with the output word $r = y_1y_2 \ldots y_k$, corresponding to the word p in the state a, but is equal to the word $y_0y_1y_2 \ldots y_{k-1}$ where y_0 is the location of the state a. The word q obtained in this way is called the *shifted output word*, corresponding to the input word in the state a.

Experiments with automata are a special part of the abstract theory of automata and have been studied by various workers, [4], [6], [7], [13], [20], [30]. Moore's work [20] seems fundamental in this respect. He was the first to formulate and solve many of the important problems in the theory of experiments. The idea is to design experiments to reveal the inner structure of automata, i.e. to determine the transfer and output functions.

In the present section we shall describe only some very important results of the theory of experiments which are connected with the remainder of this article. Departing from the established practice, we shall not confine ourselves to Moore experiments.

Two mutually equivalent (resp. Moore equivalent) states are clearly not distinguishable from one another in any experiment (resp. Moore experiment). In other words, any input word makes both these states give rise to identical results for the experiment (Moore experiment). The converse is also true: states which are indistinguishable from one another in the sense explained are certainly equivalent (resp. Moore equivalent).

The position is somewhat different for indistinguishable automata. We first give, following Moore [20] a precise definition of the corresponding concept.

DEFINITION 26. An automaton A is said to be *indistinguishable* (resp. Moore indistinguishable) form the automaton B if for each state a of the first automaton and each input word p there is a state b of the second automaton such that the experiment (resp. Moore experiment) with the input word p and the states a and b give the same result.

(Editor's note. The definition of indistinguishability given by Moore requires, in addition to this, that for each state b of B there is a state a with the properties stated at the conclusion.)

It is clear that equivalent (resp. Moore equivalent) automata will be identical, (resp. Moore identical). The converse can be shown to be untrue. Moore gave the following example to demonstrate this point. [20]. A and B are two Moore automata with the input alphabet (x, y) and the output alphabet (u, v), defined by the following table of the transfer function and locations

	u	v	u	u
	1	2	3	4
x	2	1	1	2
y	3	3	2	2

	u	v	u
	1	2	3
x	2	1	1
y	2	3	2

It is easy to see that the first automaton A is Moore equivalent to the second B. Actually, for $i = 1, 2, 3$ the state i of A is Moore indistinguishable from the state i of B. As regards the state 4 of A, it is not difficult to verify that any arbitrary Moore experiment in this state with an input word p gives the same result as the Moore experiment with the same input word in state 1, if the word p begins with the letter x, or in state 3, if the word p begins with the letter y. In spite of this, the automata are not mutually equivalent.

Thus indistinguishability (Moore indistinguishability) is a weaker condition than equivalence (Moore equivalence). Moore automata which are indistinguishable, or Moore indistinguishable, can be called weakly equivalent, or Moore weakly equivalent. This weak equivalence implies ordinary equivalence in the case of finite strongly-connected automata. This follows from a theorem proved by Moore [20] for the case of Moore indistinguishable automata.

THEOREM 41. *If a strongly-connected finite automaton A is indistinguishable (resp. Moore indistinguishable) from a finite automaton B, then for every state of A, we can find an equivalent (resp. Moore equivalent) state b in B.*

PROOF. It is sufficient to indicate the proof in the case of ordinary indistinguishability, the proof for Moore indistinguishability follows exactly the same lines. Consider an arbitrary state a of A and an arbitrary input word p. According to the theorem states of B can be found for which an experiment with the word p will give the same result as for the state a. Let $b_1, \dots, b_k$ be all such states. Denote by $M = M(a, p)$ the set of all states $b_1p, b_2p, \dots, b_kp$. For any arbitrary choice of a and p the set $M(a, p)$ is non-empty. It is easy to see that in supplementing the word p by new letters (on the right), the number of elements of the set $M(a, p)$ will not be increased. Since this set is finite, we can choose the word p so that for any arbitrary word r, the number of elements of the set $M(a, pr)$ will be equal to the number of elements of the set $M(a, p)$.

If A is strongly-connected, the word r can be chosen so that $apr = a$. Now it is easy to see that any arbitrary state b out of the set $M(a, pr)$

is indistinguishable from the state a. If this were not true there would exist a word q, for which the results of experiments on the states a and b would be different. But then there would be different results for the experiment for the input word prq in the state a and for all those states b_i $(i = 1, .., k)$, for which $b_i pr = b$. This would mean that the number of elements of the set $M(a, prq)$ would be less than the number of elements of the sets $M(a, pr)$ and $M(a, p)$ in contradiction to the choice of the word p.

Thus the state b is indistinguishable from and consequently equivalent to the state a. Since the state a can be chosen arbitrarily, the theorem is proved.

As an immediate consequence we have the following.

THEOREM 42. *If two finite strongly-connected automata are weakly equivalent (Moore weakly equivalent) then they are also mutually equivalent, (Moore equivalent).*

Let us now examine the maximum length of an experiment which is necessary to distinguish any two different states in a finite automaton. It is not difficult to see that the i-classes, discussed in §1, possess the following property: two states belong to the same i- class if and only if they are indistinguishable in any experiment of length i. Besides, it follows from §1, that in going from i to $i + 1$, the number of i-classes is increased by at least unity. For a certain value of i, the $(i + 1)$-class will coincide with the i-classes, after which the i-classes will not experience any further splitting and will include all mutually equivalent distinct states.

If there are two mutually distinguishable states the number of different 1-classes cannot be less than two. At the same time in an automaton with n states, the number of distinct i-classes cannot be more than n, whatever the value of i.

Combining these results we conclude that the mutually indistinguishable states are collected in the $(n - 1)$-classes and that any two distinguishable states can be distinguished by an experiment of length at most $(n - 1)$.

As a result we get the following theorem which is a slightly stronger form of the theorem 6 proved by Moore [20].

THEOREM 43. *In an automaton with n states, any two mutually distinguishable (Moore distinguishable) states can be distinguished by an experiment, (a Moore experiment) of length $(n - 1)$.*

Moore [20] showed that the value $(n - 1)$ in the theorem 43 cannot be decreased. To establish this, it is sufficient to examine the Moore automaton with n $(n \geqslant 3)$ states, defined by the following table of transfer functions and locations.

	v	u	u		u		u	u
	1	2	3	...	i	...	$n-1$	n
x	2	3	4	...	$i+1$	...	n	n
y	2	1	2	...	$i-1$	...	$n-2$	$n-1$

In this automaton any two states are Moore distinguishable, but the shortest Moore experiment establishing the fact that the states n and $(n - 1)$ are Moore distinguishable has the length $(n - 1)$.

Using the direct sum of automata and theorem 43, it is not difficult to show that to distinguish two distinguishable (Moore distinguishable) states in two automata with n and m states respectively, an experiment (Moore experiment) of length $m + n - 1$ will suffice.

Moore showed (for $m = n$) that this length cannot be reduced.

Moore also showed that in order to distinguish a reduced Moore automaton A with n states, m input and p output signals from all other non-isomorphic Moore reduced automata with the same input and output alphabets as A, a Moore experiment of length $n^{nm+2}p^n/n$ suffices. An analogous estimate was obtained from other considerations and in a somewhat more general case by S. Ginsburg ([6]).

As well as distinguishability between states, it is useful to introduce the concept of distinguishability of input words, (Ginsburg [7]). Two input words p and q are called *indistinguishable* in an automaton A, if for any arbitrary state a of A and for any input word r, experiments with input words pr and qr in the state a yield identical results. If the results of the experiment are not identical, the corresponding words are called *distinguishable* in the automaton A. An automaton may be said to be *distinguishable with respect to inputs*, if any two arbitrary different words can be distinguished in this automaton. Ginsburg proved the following theorem.

THEOREM 44. *Let A be a finite automaton with r states, in which any two different states are non-equivalent. Let n_i be the number of states generated by the i-th state of A. Then if any two arbitrary input words of length $(n_1n_2 \ldots n_r)^2$ are distinguishable in the automaton A, A is distinguishable with respect to inputs.*

In some cases it will be necessary to consider infinite experiments, with an infinite sequence of input signals (an input sequence). The result of such an experiment will not be a finite word but an infinite sequence of output signals (an output sequence).

We shall call a sequence, which is periodic beginning from a certain place, almost periodic. Following Ginsburg [7] the state of the automaton is said to be rational, if an infinite experiment in this state with any almost periodic input sequence gives an almost periodic output sequence. It follows from [7] that:-

THEOREM 45. *If in an automaton A any arbitrary state generates a finite set of states, then all states of A are rational.*

Automata considered in theorem 45 may be called periodic automata. All finite automata belong to this class.

Conclusion

The theory discussed in the present article is based on a conception of an automaton, in which the details of the internal construction are almost completely ignored, with the exception of the number of its states and its transfer function. Such an approach distinguishes the abstract theory of automata from the general theory of automata, in particular from the structural theory of automata. This latter studies the construction of complex automata from a collection of fixed given elementary automata.

All the most important directions in the present (1961) level of the abstract theory of automata, understood in this sense, have been included to a greater or lesser extent in this article. It remains only to mention some of the tendencies which have developed most recently.

The first is the further generalization of the notion of an abstract automaton. This tendency appeared in the work of S. Ginsburg ([7]), in which the idea of a generalized automaton or quasi-machine was introduced. By this term Ginsburg means a set of five objects - the non-empty set $\mathfrak{A}$ of states of the quasi-machine, two arbitrary abstract semi-groups $\mathfrak{B}$ and $\mathfrak{C}$, called respectively its input and output semi-groups, and two functions $\delta(a, b)$ and $\lambda(a, b)$, the transfer and output functions of the given quasi-machine. The functions δ and λ are mappings of the set $\mathfrak{A} \times \mathfrak{B}$ on the sets $\mathfrak{A}$ and $\mathfrak{C}$ respectively. For an arbitrary state $a \in \mathfrak{A}$ and any two inputs b_1 and b_2 of $\mathfrak{B}$ we have the relations $\delta(a, b_1b_2) = \delta(\delta(a, b_1), b_2)$ and $\lambda(a, b_1b_2) = \lambda(a, b_1)\lambda(\delta(a, b_1), b_2)$.

It is easy to see that by a natural extension of transfer and output functions of ordinary automata (see §1), these automata are turned into quasi-machines in which both the input and output semi-groups become free semi-groups. In this way the generalization of the idea of an automaton by Ginsburg consists in the replacement of free input and output semi-groups by arbitrary semi-groups.

To obtain the simplest interesting results for a quasi-machine, Ginsburg restricted the general conception to quasi-machines, in which the output semi-group is a semi-group with the left law of cancellation. We can transfer the ideas of the distinguishability of states, equivalence and isomorphism to these limited quasi-machines (called simply machines) with corresponding changes in nomenclature. It is easy to prove that in any class of mutually equivalent machines, there exists a unique (up to an isomorphism) reduced machine. This is the generalized form of part of our theorem 3.

The second trend consists in the introduction into the set of states of the automaton and its input and output semi-groups of a topology, order and of other structures, allowing us to consider the transfer and output functions from these points of view. (i.e. continuity, differentiability etc.) This tendency appears in the work of Schützenberger [35], who considered as the set of states of an automaton the product of a finite set and the ring of integers. The presence of an algebraical structure in the set of states allows us to consider the transfer function as a special "algebraical" function.

Finally an attempt has been made to apply the abstract theory of automata to construct an abstract theory of languages. The language is treated as an automaton with a multivalued transfer function.

References

[1] D.D. Aufenkamp and F.E. Hohn, Analysis of sequential machines, I.R.E. Trans EC-6 (1957), 276-285, translated in Matematika, 3 : 3 (1959), 129-146.

[2] D.D. Aufenkamp, Analysis of sequential machines, translated in Matematika 3 : 6 (1959), 146-158.

[3] A.W. Burks and H. Wang, The logic of automata, Journal Ass. Comp. Machines 2 (1957), 193-218, 3 (1957), 279-297.

[4] A.Sh. Blokh, On soluble problems in sequential machines, Problemi Kibernetiki - Vyp. 3 (1960), 81-88.

[5] J.R. Büchi, Weak second order arithmetic and finite automata, Zeitschr. Math. Logik und Grundlagen der Math. 6, 1, (1960) 66-92.

[6] S. Ginsburg, On the length of the smallest uniform experiment which distinguishes the terminal states of a machine, Journal Ass. Comp. Machines 5, (1958), 266-280.

[7] S. Ginsburg, Some remarks on abstract machines, Trans. Amer. Math Soc. 96, 3, (1960), 400-444.

[8] S. Ginsburg, On a method of synthesis of sequential machines, Matematika 4 : 4, (1960), 145-168.

[9] V.M. Glushkov, On a synthesis algorithm for abstract automata, Ukr. Matem. Zhurnal 12, 2, (1960), 147-156.

[10] V.M. Glushkov, On the analysis of abstract automata, Doklady Akad. Nauk U.S.S.R., No. 9, (1960), 1151-1154.

[11] V.M. Glushkov, Abstract automata and partitions of free semi-groups, Doklady Akad Nauk S.S.S.R. (1961).

[12] V.M. Glushkov, Some problems in the synthesis of automatic computers, Vychsl. Matem. i Matem. Fizika 3, (1961), 371-411.

[13] A.A. Karatsuba, On the solution of a problem in the theory of finite automata, Uspekhi Matem. Nauk 15, 3 (93) (1960), 157-159.

[14] S.C. Kleene, The representation of events in nerve-nets and finite automata. *Automata Studies*, Princeton, 1956, 3-41. Translated in Sbornik, "Avtomati" (1956), 15-67.

[15] J.M. Copy, C. Elgot, J.B. Wright, Realization of events by logical nets, Journal Ass. Comp. Machines, 5, 2, (1958), 181-196.

[16] R.F. McNaughton and H. Jamanda, Regular expressions and state graphs for automata, J.R.E. Trans. Electr. Comp. E.C.9, 1, (1960), 39-48.

[17] Yu.T. Medvedev, On classes of events which can be represented in finite automata, Sbornik "Avtomati" (1956), 385-401.

[18] F.J. Murray, Mechanisms and automata, Kibernetiki Sbornik I.L. (1960), 149-174. (Evidently a translation of "Mechanisms and Robots", Jnl. Assoc. for Computing Machinery, 2, (1955), 61-82.)

[19] G.H. Mealy, A method of synthesizing sequential circuits, Bell System Tech. Journal, 34, (1955), 1045-1079.

[20] E.F. Moore, Gedanken-experiments on sequential machines, *Automata Studies*, Princeton, (1956) 129-153, translated in Sb. Avtomati, (1956), 179-212.

[21] D.S. Netherwood, Minimal sequential machines, J.R.E. Trans Electr. Comp. E.C.8, 3, (1959), 339-345.

[22] M.C. Paull and S.H. Unger, Minimizing the number of states in incompletely specified searching functions, J.R.E. Trans. Electr. Comp. E.C.8, 3, (1959), 356-367.

[23] M.O. Rabin and D. Scott, Finite automata and their decision problems, I.B.M. Research Journal, 3, (1959), 114-117.

[24] G.N. Raney, Sequential functions, Journal Ass. Comp. Machines, 5, 2, (1958), 177-180.

[25] S. Seshu, Mathematical models for sequential machines, J.R.E. Nat. Convention Record, 7, 2, (1959), 4-16.

[26] L.A. Skornyakov, Nerve systems, Uspekhi Matem. Nauk 13 vyp. 3, (81), (1958), 233-234.

[27] L.A. Skornyakov, On a class of automata (nerve systems), Problemi Kibernetiki vyp. 4, (1960), 23-36.

[28] B.A. Trakhtenbrot, The synthesis of logical nets using the calculus of single-valued predicates, Doklady Akad. Nauk U.S.S.R. 118, 4, (1958).

[29] D.A. Huffman, The synthesis of sequential switching circuits, Journal Franklin Inst. 257, 3, (1954), 161-190.

[30] S. Huzino, On some sequential machines and experiments, Mem. Fac. Sci. Kyusyu Univ. series A 12, 2, (1958), 136-158.

[31] S. Huzino, Reduction theorems on sequential machines and experiments, Mem. Fac. Sci. Kyusyu Univ. series A12, 2, (1958), 159-172.

[32] S. Huzino, On the existence of a Sheffer stroke class in sequential machines, Mem. Fac. Sci. Kyusyu Univ. series A13, (1959), 52-68.

[33] S. Huzino, Some properties of convulution machines and σ-composite machines, Mem. Fac. Sci. Kyusyu Univ. series A13, (1959), 69-83.

[34] J.C. Shepherdson, The reduction of two-way automata to one-way automata, I.B.M. Journal Res. and Rev. 3, (1959), 198-199.

[35] M.P. Schützenberger, Un problème de la theorie des automates, Semin. Dubreuil - Pisot, 13^e année No. 3 (1959/60).

[36] A.A. Letichevskii, On the synthesis of finite automata, Doklady Akad. Nauk U.S.S.R. No. 2 (1961), 139-141.

[37] A.A. Letichevskii, The conditions of completeness for finite automata, Vychisl. Matem. i Matem. Fizika No. 4 (1961).

[38] Yu.I. Sorkin, The algebra of automata, Problemi Kibernetiki (1961).

Translated by J.M. Jackson.