Ivan M. Havel

Regular expressions over generalized alphabet and design of logical nets

Kybernetika, Vol. 4 (1968), No. 6, (516)--537

Persistent URL: http://dml.cz/dmlcz/125463

KYBERNETIKA ČÍSLO 6, ROČNÍK 4/1968

Regular Expressions over Generalized Alphabet and Design of Logical Nets

IVAN M. HAVEL

Regular expressions over the so called generalized alphabet are introduced. The symbols of this alphabet are interpreted as sets of input states of a finite automaton instead of these states alone. The symbols may correspond e.g. to signals on individual input channels of a multiple-input device. A synthesis procedure using derivatives of expressions is shown. This procedure leads from regular expressions over generalized alphabet to transition tables of abstract automata or directly to diagrams of corresponding logical nets. These nets have a special character similar to that of Yamada's disjunctively linear logical nets and allowing the composition of nets in accordance with basic Kleene's operations.

1. INTRODUCTION

Because of its simplicity and algebraic character, the language of regular expressions is preferable to other formalized languages describing the behavior of finite automata. From the designer's point of view this language has, however, some disadvantages, which, even though not fundamental, make the application of regular expressions often somewhat cumbersome. One of these disadvanteges is the fact that with multiple-input problems an abstract alphabet of input states is needed in all cases. In some of them the description in terms of signals on individual input terminals would be much more convenient. (In this point regular expressions differ e.g. from the description by means of formulas of symbolic logic.)

One of the aims of this paper is to show that the application of an alphabet containing symbols for signals appearing on the individual input terminals, or for any other elementary input situations, not only is adequate, but yields also some novel views on the design procedure and on the transition to the structural realization by means of logical nets. The starting point will be the generalized input alphabet which has been previously introduced by author in [7]. (Some sort of generalized alphabet was also introduced in [6] for the special purpose of minimizing flow tables.)

2. NOTATION AND PRELIMINARIES

Well-known notions and results of the theory of finite automata and of the algebra of regular events are used, in particular the Brzozowski's method of finite automata synthesis by means of derivatives of regular expressions [3]. Some basic concepts and notations are surveyed below.

Let $X = \{x_1, \ldots, x_m\}$ be a finite nonempty set of symbols (the *alphabet*); let us denote X^* the set of all words over X. Algebraically X^* is a *free monoid* with an associative operation (*concatenation*; the dot is usually omitted) and Λ, the *word of zero length*, as the identity; the subsets of X^*, i.e. the elements of the power set $\mathcal{P}(X^*)$, are called *events over* X. Let us introduce Kleene's operations $\cup$ (*union* of sets), $\cdot$ (*concatenation* of sets) and $*$ (*star operation*) in the set of events.

The *algebra of regular events* or the *Kleenean algebra* $\mathcal{K}(X)$ is defined by means of Kleene's operations and of the set of generators $\{\{x_1\}, \ldots, \{x_m\}, \emptyset\}$. It is plausible to distinguish the algebra of regular events from the *language of regular expressions*. The latter is defined as a free algebra $L(X) = \langle X, \cup, \cdot, *, \emptyset \rangle$ where $\cup$ and $\cdot$ are associative binary operations, $*$ is unary and $\emptyset$ nullary operation. The relation of $L(X)$ to $\mathcal{K}(X)$ is determined by the *interpretation* of $L(X)$, which is a mapping $|| : L(X) \to$ $\to \mathcal{P}(X^*)$ for which $|x_i| = \{x_i\}$, $i = 1, \ldots, m$ and symbols $\cup$, $\cdot$, and $*$ denote corresponding Kleene's operations. The mapping $||$ yields a relation of equivalence in $L(X)$, denoted by $=$. It is convenient to include the symbols Λ $(= \emptyset^*)$ and X $(= x_1 \cup \ldots \cup x_m)$ also as symbols of the language. The main purpose of the language $L(X)$ is to make possible to describe events of $\mathcal{K}(X)$ which may be generally infinite sets (whereas the elements of $L(X)$ are only finite strings of symbols).

We shall also use matrices over $L(X)$. Operations on such matrices are defined similarly to those on matrices over number fields where addition and multiplication (of entries) is replaced by union and concatenation.

For each $x_i \in X$ we define inductively a *derivative with respect to* x_i [3] as the unary operation $\partial_i : L(X) \to L(X)$ as follows:

(1) $\forall x \in X, \quad \partial_i x = \begin{cases} \Lambda & \text{if } x = x_i; \\ \emptyset & \text{otherwise;} \end{cases}$

(2) $\partial_i \emptyset = \emptyset$;

(3) $\partial_i(\alpha \cup \beta) = \partial_i\alpha \cup \partial_i\beta$;

(4) $\partial_i(\alpha\beta) \quad = (\partial_i\alpha)\, \beta \cup \varrho(\alpha)\, \partial_i\beta$;

(5) $\partial_i\alpha^* \quad = (\partial_i\alpha)\, \alpha^*$;

where

$$\varrho(\alpha) = \begin{cases} \Lambda & \text{if } \Lambda \in |\alpha| \\ \emptyset & \text{otherwise}. \end{cases}$$

The derivative denotes the event $|\partial_i\alpha| = \{w \in X^* \mid x_i w \in |\alpha|\}$ in $\mathcal{K}(X)$. The *derivative closure* $D(\alpha)$ of the expression α is defined as the set of expressions possessing the following properties:

(1) $\alpha \in D(\alpha)$;

(2) $\forall \beta \in D(\alpha) , \quad \forall x_i \in X , \quad \partial_i \beta \in D(\alpha)$.

$D(\alpha)$ can be constructed by finding repeated derivatives of α with respect to all $x_i \in X$ starting with α itself. If only mutually nonequivalent expressions are considered, the closure $D_0(\alpha)$ which is finite and has minimal cardinality is obtained.

A *finite automaton over the alphabet* X is defined as a quintuple $\mathscr{A} = \langle X, S, s_1, \delta, F\rangle$, where S is a finite nonempty *set of states*, $s_1 \in S$ the *initial state*, $\delta : S \times X \to S$ the *transition function* and $F \subseteq S$ the set of final states. Occationaly we shall use the term "abstract automaton" in order to emphasize the cases when the sets X and S are abstract, noninterpreted sets.

If the transition function is extended as usual to the mapping $\delta : S \times X^* \to S$, one can attribute to any finite automaton $\mathscr{A}$ a set

$$T(\mathscr{A}) = \{w \in X^* \mid \delta(s_1, w) \in F\} ,$$

called the *event recognized by the automaton* $\mathscr{A}$; individual words of this event are said to be accepted by the automaton. By means of the event $T(\mathscr{A})$ we define the *behavior* of $\mathscr{A}$. Two automata $\mathscr{A}_1$, and $\mathscr{A}_2$ are equivalent iff* $T(\mathscr{A}_1) = T(\mathscr{A}_2)$. We may formulate the well-known *Kleene's theorem* in the following way:

$$\forall A \in \mathscr{P}(X^*) , \quad A \in \mathscr{K}(\times) \quad \text{iff} \quad \exists \mathscr{A}, \, A = T(\mathscr{A}) .$$

A procedure leading from a given expression $\alpha \in L(X)$ to an automaten $\mathscr{A}_\alpha$ (i.e. to its transition table) such, that $T(\mathscr{A}_\alpha) = |\alpha|$, will be called an *abstract synthesis*. In this paper we are concerned with Brzozowski's abstract synthesis [3]. It consists in the construction of an automaton $\mathscr{A}_\alpha = \langle X, S, s_1, \delta, F\rangle$, where

(1) $S = D_0(\alpha)$ (or some other finite closure of α);

(2) $s_1 = \alpha$;

(3) $\forall \beta \in S, \ \forall x_i \in X, \ \delta(\beta, x_i) = \partial_i \beta$;

(4) $F = \{\beta \in S \mid \varrho(\beta) = \Lambda\}$.

A special case of the finite automaton is a *binary automaton* $\mathscr{A}^{(2)} = \langle \boldsymbol{X}, \boldsymbol{S}, \boldsymbol{s}_1, \boldsymbol{\delta}, \boldsymbol{F}\rangle$ with $\boldsymbol{X}$ and $\boldsymbol{S}$ interpreted as sets of binary vectors $\boldsymbol{X} \subseteq \{0, 1\}^p$, $\boldsymbol{S} \subseteq \{0, 1\}^q$ where p, q are some integers. (Binary vectors are denoted by bold italics.) It is useful to replace the final set $\boldsymbol{F}$ by the *output function* $\lambda : \boldsymbol{S} \to \{0, 1\}$ (we confine our attention to a single binary output) where $\lambda(\boldsymbol{s}) = 1$ iff $\boldsymbol{s} \in \boldsymbol{F}$. (The functions $\boldsymbol{\delta}$ and λ may be defined by means of equations over Boolean expressions.)

* I.e. if and only if.

Input and *state assignments* are two mappings $\mathbf{f} : X \to \{0, 1\}^p$ and $\mathbf{g} : S \to \{0, 1\}^q$ which determine some suitable binary automaten $\mathscr{A}^{(2)}$ to a given abstract automaton $\mathscr{A}$ ($\mathbf{g}$ is generally admitted to be many-valued).

A procedure which applies to an abstract automaton $\mathscr{A}$ and yields a logical net $\mathscr{N}$ realizing $\mathscr{A}$ is called a *structural synthesis.*

Here an abstract *logical net* $\mathscr{N}$ is given by its *structure* and *internal behavior.* The structure consists of *elements* of certain type (in our case the idealized elements OR-gate, AND-gate, inverter, and unit delay), which are properly connected by means o *connections.* Several *input terminals* and *output terminals* (in our case only one output terminal) also belong to the net. The internal behavior of the net can be described as follows: At first we associate the corresponding Boolean functions with elements of the given types. Then we define the function ϑ which assigns each connection and each time t ($t = 0, 1, \ldots$) some *value* from $\{0, 1\}$ (the *signal*), so that the appropriate boolean relations are fulfilled for all elements of $\mathscr{N}$. It is possible to show that to the entire knowledge of the function ϑ at any time t it is sufficient to know its values only (1) on input terminals (the values of *input variables*) at all $t' \leqq t$ and (2) on some selected connections (the values of internal variables) at $t = 0$. Then the class of all admitted functions ϑ determines the *internal behavior* of the given net $\mathscr{N}$.

If interested only in the value of ϑ on the output terminal (the *output variable*) in dependence on values of input and internal variables, we speak about *external behavior* or shortly *behavior* of the net.

By neglecting the structure of a net we arrive at a concept of (generally noninitial) *binary automaton describing the net.* Vectors of values of input and internal variables (shortly called *input* and *internal states*) constitute the set $\boldsymbol{X}$ and $\boldsymbol{S}$ respectively. The functions δ and λ can be derived from the internal behavior of the net.

We have reviewed those principal concepts which will be used in the further text. More thorough investigation of these concepts may be found in special literature.

3. THE GENERALIZED INPUT ALPHABET

When describing the behavior of finite automata by means of regular expressions, the elementary symbols are those of the input alphabet X. There are, however, many cases when some other objects would be more convenient to be taken as elementar. One example of this type is the case mentioned previously when the behavior may be described in terms of signals on individual input terminals. As other example let us assume that the input alphabet is partitioned to several classes of symbols (e.g. even – odd numbers in arithmetic interpretation of the alphabet) and that the designed automaton is to distinguish only symbols of the different classes.

Our first step will be to generalize suitably the concept of input alphabet to include all similar cases as well. The most convenient means of doing this proves to be the power set $\mathscr{P}(X)$ (the set of all subsets of X). The disadvantage of large cardinality of this set will not be serious becouse of the possibility of using only a small part of it in particular cases.

Definition 1. Let $X = \{x_1, \ldots, x_m\}$ be an alphabet and let $\Xi = \{\xi_1, \ldots, \xi_\mu\}$ with $\mu \leqq 2^m$ be some set of symbols. The set is called the *generalized alphabet* to X if an injective* mapping $\varphi : \Xi \to \mathscr{P}(X)$ is given.

The mapping φ, which is substantial for applications of the generalized alphabet, is determined by the so called *transform matrix*

$$\Delta = [\Delta_{\lambda i}] \; ; \quad \lambda = 1, \ldots, \mu \; ; \quad i = 1, \ldots, m$$

where

$$\Delta_{\lambda i} = \begin{cases} \Lambda & \text{if } x_i \in \varphi(\xi_\lambda) \\ \emptyset & \text{otherwise} . \end{cases}$$

Example 1. The input of designed automaton consists of two independent binary channels; the alphabet X has four elements; $x_1 = (0, 0)$, $x_2 = (0, 1)$, $x_3 = (1, 0)$, $x_4 = (1, 1)$. Let symbols ξ_1 and ξ_2 denote the occurences of 1 on the first and second channel respectively. Then the alphabet $\Xi = \{\xi_1, \xi_2\}$ is a generalized alphabet to x with the transform matrix

$$\Delta = \begin{bmatrix} \emptyset & \emptyset & \Lambda & \Lambda \\ \emptyset & \Lambda & \emptyset & \Lambda \end{bmatrix} .$$

Example 2. The alphabet $x = \{x_1, x_2, \ldots, x_8\}$ represents numbers $0, 1, \ldots, 7$. Let symbols ξ_1 and ξ_2 denote the numbers nondivisible by 2 and 3 respectively. Then the alphabet $\Xi = \{\xi_1, \xi_2\}$ is a generalized alphabet to X with the transform matrix

$$\Delta = \begin{bmatrix} \Lambda & \Lambda & \emptyset & \Lambda & \emptyset & \Lambda & \emptyset & \Lambda \\ \Lambda & \Lambda & \Lambda & \emptyset & \Lambda & \Lambda & \emptyset & \Lambda \end{bmatrix} .$$

We are able to define the corresponding language of regular expressions $L(\Xi) = \langle \Xi, \cup, \cdot, *, \emptyset \rangle$ for the generalized alphabet Ξ in the same way as it was done for X. The symbols $\emptyset$ and $\Lambda = \emptyset^*$ are assumed to be common for both languages.

To make it possible for the expressions of $L(\Xi)$ to denote the events over X, a translation from $L(\Xi)$ to $L(X)$ is needed.

Definition 2. The *translation from* $L(\Xi)$ *to* $L(X)$ is defined as a mapping $\tau : L(\Xi) \to L(X)$ in the following way:

(1) $\tau(\xi_\lambda) = x_1 \Delta_{\lambda 1} \cup x_2 \Delta_{\lambda 2} \cup \ldots \cup x_m \Delta_{\lambda m}$; $\lambda = 1, \ldots, \mu$;

(2) τ preserves the operations $\cup, \cdot, *, \emptyset$ and their ordering.

* I.e. one-one into.

Remark. If α_i $(i = 1, \dots, k)$ are expressions of a given language of regular expressions, we will use the notation $\bigcup_{i=1}^{k} \alpha_i$ instead of $\alpha_1 \cup \alpha_2 \cup \dots \cup \alpha_k$. (The reader is recommended to keep in mind that the symbol $\cup$ is in this paper used in two different meanings: to denote the union of sets and as the symbol of the language of regular expressions.)

Definition 3. A generalized alphabet Ξ is said to have the *covering property* if

$$\forall x \in X \quad \exists \xi \in \Xi\,, \quad x \in \varphi(\xi)\,.$$

The generalized alphabet has to posses this property e.g. if we want tu use the symbols Ξ and Ξ^* in the expressions of $L(\Xi)$ (similarly as the symbol X and X^* in $L(X)$) to denote the events: "All words of the length 1" and "All words" respectively.

The following result is an immediate consequence of the covering property:

$$\tau(\Xi) = \tau\Big(\bigcup_{\lambda=1}^{\mu} \xi_\lambda\Big) = \bigcup_{\lambda} \bigcup_{i=1}^{m} x_i \Delta_{\lambda i} = \bigcup_{i}\Big(x_i \bigcup_{\lambda} \Delta_{\lambda i}\Big) = \bigcup_{i} x_i = X\,.$$

From Definition 2 we see that the transition from given $\alpha \in L(\Xi)$ to $\tau(\alpha) \in L(X)$ consists only in replacing each ξ which occurs in the expression α by the corresponding expression $\bigcup_{j \in J} x_j$, $J \subseteq \{1, \dots, m\}$ (or, as the case may be, in replacing of Ξ by X). The operations $\cup, \cdot, *$ and the constants $\emptyset$, Λ remain the same.

The following lemma that establishes a relationship of derivatives in $L(X)$ and $L(\Xi)$ will be needed later.

Lemma 1. *For any* $\alpha \in L(\Xi)$

$$\partial_i\, \tau(\alpha) = \tau\Big(\bigcup_{\lambda=1}^{\mu} \partial_\lambda \alpha \Delta_{\lambda i}\Big)\,, \quad i = 1, \dots, m\,,$$

where ∂_λ *and* ∂_i *are derivatives in* $L(\Xi)$ and $L(X)$ *respectively.*

Proof. Since

$$\alpha = \bigcup_{\lambda} \xi_\lambda\, \partial_\lambda \alpha \cup \varrho(\alpha)$$

(see [3]) we have

$$\tau(\alpha) = \tau\Big(\bigcup_{\lambda} \xi_\lambda\, \partial_\lambda \alpha \cup \varrho(\alpha)\Big) = \bigcup_{\lambda} \tau(\xi_\lambda)\, \tau(\partial_\lambda \alpha) \cup \varrho(\alpha) = \bigcup_{\lambda}\Big(\bigcup_{i=1}^{m} x_i \Delta_{\lambda i}\Big)\, \tau(\partial_\lambda \alpha) \cup \varrho(\alpha)$$

and thus

$$\partial_i \tau(\alpha) = \bigcup_{\lambda} \Delta_{\lambda i}\, \tau(\partial_\lambda \alpha) = \tau\Big(\bigcup_{\lambda} \partial_\lambda \alpha \Delta_{\lambda i}\Big)\,,$$

Q.E.D.

The following lemma is an immediate consequence of Definition 2.

Lemma 2. *If $\alpha = \beta$ in $L(\Xi)$then $\tau(\alpha) = \tau(\beta)$ in $L(X)$ (note that '$=$' means only equivalence).*

The converse, however, is not, in general, true.

We conclude this section with some comments about the features of the generalized alphabet in practical cases. Usually the alphabet Ξ follows from the character of the problem given, and it must contain symbols for all situations which are to appear in the design requirements. If the absence of some of such situations is required, or a simultaneous presence of two different situations (as e.g. the occurence of 1 on both channels in Example 1), it turns out to be necessary to introduce a new independent symbol of the alphabet Ξ. (The reason being that the results of this paper are not generally applicable to a language with complement and intersection.) In some special cases Ξ must posses the covering property (Definition 3). The case that $\varphi(\xi) = \emptyset$ for some ξ is not generally excluded. It causes the substitution of $\emptyset$ wherever ξ appears in a given expression and it corresponds well to a situation physically not realizable.

4. SYNTHESIS OF FINITE AUTOMATON

In this section we shall discuss the synthesis of a finite automaton, described by an expression over the generalized alphabet. Solid arrows in the following diagram illustrate the direction in which we shall proceed:

$$\begin{tikzcd}
\alpha \in L(\Xi) \arrow[r, dashed, "\tau"] \arrow[d, "\text{Synthesis in } L(\Xi)"'] & \tau(\alpha) \in L(X) \arrow[d, dashed, "\text{Synshesis in } L(X)"] \\
\mathscr{A}_\alpha \arrow[r, "\pi"] & \mathscr{A}_{\tau(\alpha)}
\end{tikzcd}$$

Here $\mathscr{A}_\alpha$ is an auxiliary automaton over Ξ, and $\mathscr{A}_{\tau(\alpha)}$ is a desired automaton over X.

The aim of this section is to show that this direction yields a procedure which is equally applicable (i.e. it leads to an equivalent result) as the synthesis via the language $L(X)$ (dashed arrows).

Let the behavior of a finite automaton be given by a regular expression $\alpha \in L(\Xi)$. In order to synthesize the automaton via $L(X)$ one must take the following way:

(1) Replace the expression α by its translation $\tau(\alpha)$;

(2) Perform the usual synthesis procedure in the language $L(X)$.

(The result is the transition table of the automaton $\mathscr{A}_{\tau(\alpha)}$).

As already mentioned, the basis of the synthesis of $\mathscr{A}_{\tau(\alpha)}$ in $L(X)$ is the construction of a derivative closure $D(\tau(\alpha))$ by finding all successive derivatives of certain expressions. Lemma 1 permits to express these derivatives by means of derivatives of appropriate expressions in $L(\Xi)$. It is therefore feasible to construct first a derivative closure

$D(\alpha)$ in the language $L(\Xi)$ (this closure is basically the auxiliary automaton $\mathscr{A}_\alpha$ over Ξ) and only after having done this to transfer to the resulting proper automaton $\mathscr{A}_{\tau(\alpha)}$.

Definition 4. Let $\alpha \in L(\Xi)$ and let $\mathscr{A}_{\tau(\alpha)}$ be an automaton over X for which $T(\mathscr{A}_{\tau(\alpha)}) = |\tau(\alpha)|$. Then the automaton $\mathscr{A}_\alpha$ over Ξ is called the *auxiliary automaton* (to $\mathscr{A}_{\tau(\alpha)}$) if $T(\mathscr{A}_\alpha) = |\alpha|$.

To avoid confusion we write automata over Ξ as $\mathscr{A} = \langle \Xi, \Sigma, \sigma_1, \delta'_1, \Phi \rangle$, using Greek letters Ξ, Σ, $\sigma_\varkappa$, Φ instead of X, S, s_i, F. The automata over X will be sometimes called *proper automata.*

Remark. Some precaution in interpretation of just defined auxiliary automaton is desirable. When calling it an "automaton" we have to consider the alphabet Ξ as abstract set of symbols with no interpretation. Only under this condition we can (and we do) speak about the synthesis procedure in $L(\Xi)$ being performed according to the same rules as the synthesis in $L(X)$. However, being considered as an abstract automaton, $\mathscr{A}_\alpha$ has very little in common with the corresponding proper automaton $\mathscr{A}_{\tau(\alpha)}$ (having e.g. different input alphabet and generally different – larger or smaller – number of states).

Nevertheless, there is some informal way to describe the connection of $\mathscr{A}_\alpha$ with $\mathscr{A}_{\tau(\alpha)}$. If it were possible for more symbols of Ξ to "occur simultaneously" on the input of $\mathscr{A}_\alpha$ and if then $\mathscr{A}_\alpha$ were able to "occur simultaneously" in several different states, it might display the "same behavior" as $\mathscr{A}_{\tau(\alpha)}$ under following two assumptions:

(1) Simultaneous occurence of all $\xi \in \Xi$ for which $x \in \varphi(\xi)$ corresponds to each $x \in X$.

(2) $\mathscr{A}_\alpha$ accepts just those words from X^* which (as sets of words in accordance with (1)) cause the transition to some set of simultaneous states, containing at least one final state from Φ.

Now we shall show a procedure which applies to a given auxiliary automaton $\mathscr{A}$ over Ξ and yields certain automaton $\pi(\mathscr{A})$ over X. We shall see below (Theorem 2) that the latter is equivalent to the desired proper automaton constructed in $L(X)$.

Procedure 1. (We begin with an automaton $\mathscr{A} = \langle \Xi, \Sigma, \sigma_1, \delta', \Phi \rangle$ obtained as a result of the synthesis by means of derivatives in $L(\Xi)$; it is described by means of a transition table $\mathbf{\Gamma}_1$ having μ columns denoted by $\xi_1, \ldots, \xi_\mu (\in \Xi)$ and having ν rows denoted by $\sigma_1, \ldots, \sigma_\nu (\in \Sigma)$. In addition to it a set $\Phi \subseteq \Sigma$ is given. We shall consider the table $\mathbf{\Gamma}_1$ to be a $\nu \times \mu$-matrix over $L(\Xi)$.) The transformation of an automaton $\mathscr{A}$ to $\pi(\mathscr{A})$ consists of two parts:

Part 1. Multiply the matrix $\mathbf{\Gamma}_1$ from the right by the transform matrix $\mathbf{\Delta}$.

(The result is a $v \times m$-matrix or table $\mathbf{\Gamma}_2 = \mathbf{\Gamma}_1 \,.\, \mathbf{\Delta}$. Denote its rows again by $\sigma_1, \ldots, \sigma_v$ and denote the columns by $x_1, \ldots, x_m$. Entries in the table $\mathbf{\Gamma}_2$ are expressions of the type $\bigcup_{\gamma \in I} \sigma_\gamma$; $I \subseteq \{1, \ldots, v\}$. Here we are justified to use the symbol $\cup$ because σ_γ's may be interpreted as expressions of $L(\Xi)$. However, we do not presuppose the familiarity with their structure and thus, comparing the entries of the table $\mathbf{\Gamma}_2$ and also $\mathbf{\Gamma}_3$, we will use only the associative and commutative laws for $\cup$ and the rule $\alpha \cup \emptyset = \alpha$.)

Part 2. consists in constructing the transition table $\mathbf{\Gamma}_3$ of the automaton $\pi(\mathscr{A})$ (with m columns and $n \leqq 2^v$ rows) by the following iterative procedure:

1. Denote the first row ($k = 1$) of the table $\mathbf{\Gamma}_3$ by σ_1 and enter the first row of $\mathbf{\Gamma}_2$ into it. Put $k = 2$.

2. In the filled up part of the table $\mathbf{\Gamma}_3$ find the first expression, which has not yet denoted any row of $\mathbf{\Gamma}_3$ and denote by it the k-th row. If there is no such expression go to step 4.

3. Let the k-th row of the table $\mathbf{\Gamma}_3$ be denoted by the expression $\bigcup_{\lambda \in I} \sigma_\gamma$. Join columnwise all rows of $\mathbf{\Gamma}_2$ denoted by some symbol σ_γ, $\gamma \in I$ by the operation $\cup$ and enter the resulting expressions into the k-th row of $\mathbf{\Gamma}_3$. Replace k by $k + 1$. Go to step 2.

4. Among the expressions denoting the rows of $\mathbf{\Gamma}_3$ find those in which at least one symbol belonging to Φ occurs.

5. Use new symbols for the entries of the table $\mathbf{\Gamma}_3$ (e.g. the symbols $s_1, \ldots, s_n$; s_1 for σ_1). Stop.

(The table $\mathbf{\Gamma}_3$ is the transition table of the automaton $\pi(\mathscr{A})$; its entries are interpreted as the states, the first row corresponds to the initial state and the final states are determined in step 4).

Remark. When considering equivalences, we did not take account of an internal structure of expressions designated by $\sigma_\varkappa$; we also neglected the relationships among the individual symbols in Ξ given by the mapping φ. Thus the minimility of the resulting automaton $\pi(\mathscr{A})$ in not guaranteed even if the auxiliary automaton $\mathscr{A}$ is reduced. (The finiteness of $\pi(\mathscr{A})$ is, however, guaranteed – cf. [3]). When applying the procedure, it is often advantageous to take the structure of the expressions into the consideration, especially when e.g. $\sigma_\varkappa = \emptyset$ or $\sigma_\varkappa = \Xi^*$ for some $\varkappa$. In many cases it is worth while to pay attention to the set inclusion of events represented by states of the auxiliary automaton. Such information is very useful, because if $|\sigma_1| \subseteq |\sigma_2|$ then $\sigma_1 \cup \sigma_2 = \sigma_2$.

New, we present an example illustrating the above procedure.

Example 3. Let the generalized alphabet Ξ be given by the transform matrix

$$\mathbf{\Delta} = \begin{bmatrix} \emptyset & \emptyset & \Lambda & \Lambda \\ \emptyset & \emptyset & \Lambda & \Lambda \end{bmatrix}$$

(see Example 1).

The behaviour of the automaton is described by the expression in $L(\Xi)$:

$$\alpha = [(\xi_1 \cup \xi_2)\, \xi_2^* \xi_1]^* \, (\xi_1 \cup \xi_2)\, \xi_2^* \, .$$

(Compare the corresponding expression in $L(X)$:

$$\tau(\alpha) = [(x_2 \cup x_3 \cup x_4)\,(x_3 \cup x_4)^*\,(x_2 \cup x_4)]^* \, . \, (x_2 \cup x_3 \cup x_4)\,(x_3 \cup x_4)^* .$$

The synthesis of $\mathscr{A}_\alpha$ using derivatives in $L(\Xi)$ proceeds as follows:

$$\begin{aligned} \alpha &= \sigma_1 \, , \\ \partial_1 \sigma_1 &= \xi_2^* \xi_1 \alpha \cup \xi_2^* = \sigma_2 \qquad (\varrho(\sigma_2) = \Lambda) \, , \\ \partial_2 \sigma_1 &= \sigma_2 \, , \\ \partial_1 \sigma_2 &= \alpha = \sigma_1 \, , \\ \partial_2 \sigma_2 &= \sigma_2 \, , \end{aligned}$$

thus the transition table $\mathbf{\Gamma}_1$ of $\mathscr{A}_\alpha$ is

	ξ_1	ξ_2
σ_1	σ_2	σ_2
σ_2	σ_1	σ_2

and $\Phi = \{\sigma_2\}$.

Further step shown is the transformation of $\mathscr{A}_\alpha$ to $\pi(\mathscr{A}_\alpha)$ by Procedure 1.

(1) By multiplication

$$\mathbf{\Gamma}_1 \, . \, \mathbf{\Delta} = \begin{bmatrix} \sigma_2 & \sigma_2 \\ \sigma_1 & \sigma_2 \end{bmatrix} . \begin{bmatrix} \emptyset & \Lambda & \emptyset & \Lambda \\ \emptyset & \emptyset & \Lambda & \Lambda \end{bmatrix}$$

we obtain the table $\mathbf{\Gamma}_2$:

	x_1	x_2	x_3	x_4
σ_1	$\emptyset$	σ_2	σ_2	σ_2
σ_2	$\emptyset$	σ_1	σ_2	$\sigma_1 \cup \sigma_2$

(2) The second part of Procedure 1 yields the table $\mathbf{\Gamma}_3$:

	x_1	x_2	x_3	x_4	
σ_1	$\emptyset$	σ_2	σ_2	σ_2	(1st row of Γ_2)
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	($\emptyset$ contains no symbol)
σ_2	$\emptyset$	σ_1	σ_2	$\sigma_1 \cup \sigma_2$	(2nd row of Γ_2)
$\sigma_1 \cup \sigma_2$	$\emptyset$	$\sigma_1 \cup \sigma_2$	σ_2	$\sigma_1 \cup \sigma_2$	(1st $\cup$ 2nd row of Γ_2).

This table is already the transition table of $\pi(\mathscr{A}_\alpha)$. We use new symbols:

	x_1	x_2	x_3	x_4
s_1	s_2	s_3	s_3	s_3
s_2	s_2	s_2	s_2	s_2
s_3	s_2	s_1	s_3	s_4
s_4	s_2	s_4	s_3	s_4

The final set is $F = \{s_3, s_4\}$. In this case the $\pi(\mathscr{A}_\alpha)$ is already reduced.

Theorem 1. *Let α be an expression of $L(\Xi)$ and $\tau(\alpha)$ its translation in $L(X)$. Let $\mathscr{A}_\alpha$* and *$\mathscr{A}_{\tau(\alpha)}$ be two automata over Ξ and X, corresponding to α and $\tau(\alpha)$ respectively. Then $\mathscr{A}_{\tau(\alpha)}$ is equivalent to $\pi(\mathscr{A}_\alpha)$, constructed by Procedure 1 from $\mathscr{A}_\alpha$.*

Proof. We shall prove that the synthesis of $\pi(\mathscr{A}_\alpha)$ from $\mathscr{A}_\alpha$ by Procedure 1 in some sense simulates the synthesis of $\mathscr{A}_{\tau(\alpha)}$ by means of derivatives in $L(X)$.

First, we briefly recapitulate the latter. It is useful to construct the derivative closure $D(\tau(\alpha))$ (we do not confine our attention to $D_0(\tau(\alpha))$ already in tabular form. Resulting table is the transition table $\mathbf{\Gamma}$ of $\mathscr{A}_{\tau(\alpha)}$. We proceed as follows:

(1) We denote the first row of $\mathbf{\Gamma}$ by $s_1 = \tau(\alpha)$. The entries s_{1i} of this row will be computed by derivatives: $s_{1i} = \partial_i s_1$.

(2) Assume that we have already filled up the $k - 1$ first rows. We find the first expression s_{li} $(l < k)$ which is not equivalent to any of those which denote these rows. We put $s_k = s_{li}$ and denote the k-th row by s_k. We compute its entries $s_{ki} = \partial_i s_k$. So we proceed further. Since $D(\tau(\alpha))$ is finite we finish with some $k = n'$. (There are no more entries nonequivalent to any s_k.) The result is $n' \times m$-table $\mathbf{\Gamma} = [s_{ki}]$.

Quite similar to this is the construction of $D(\alpha)$ over $L(\Xi)$ (instead of s we write σ; $\sigma_1 = \alpha$). Here the result is $\nu \times \mu$-table $\mathbf{\Gamma}_1 = [\sigma_{\varkappa\lambda}]$.

In the first part of Procedure 1 $\mathbf{\Gamma}_1$ is multiplied by $\mathbf{\Delta}$, which gives the $\nu \times m$-table $\mathbf{\Gamma}_2 = [\sigma'_{\varkappa i}]$. Its enries are

$$\sigma'_{\varkappa i} = \bigcup_{\lambda=1}^{\mu} \sigma_{\varkappa\lambda}\Delta_{\lambda i} = \bigcup_{\lambda} \partial_\lambda \sigma_\varkappa \Delta_{\lambda i} .$$

Now, in the second part of Procedure 1, the $n \times m$-table $\mathbf{\Gamma}_3 = [\sigma''_{ki}]$ is constructed. Its first row is the same as that of $\mathbf{\Gamma}_1$ (it is denoted by $\sigma''_1 = \sigma'_1 = \sigma_1$):

$$\sigma''_{1i} = \sigma'_{1i} = \bigcup_{\lambda=1}^{\mu} \partial_\lambda \sigma_1 \Delta_{\lambda i} = \bigcup_{\lambda} \partial_\lambda \alpha \Delta_{\lambda i} .$$

Then by Lemma 1

$$\tau(\sigma'_{1i}) = \partial_i \tau(\alpha) = \partial_i s_1$$

and thus it corresponds to the first row of $\mathbf{\Gamma}$. The remaining rows are constructed quite simularly as in the case of $\mathbf{\Gamma}$ with the exception that, instead of computing s_{ki} by derivation of s_k, we proceed as described in step 3 of Procedure 1 (Part 2).

Let σ''_k denote the k-th row of $\mathbf{\Gamma}_3$. We know that $\sigma''_k = \bigcup_{\gamma \in I_k} \sigma_\gamma$ $(I_k \subseteq \{1, \ldots, \mu\}$; $\sigma_\gamma \in D(\alpha))$. The k-th row then contains expressions

$$\sigma''_{ki} = \bigcup_{\gamma \in I_k} \sigma'_{\gamma i} \quad (i = 1, \ldots, m)$$

(see Step 3 of Procedure 1). We may write

$$\tau(\bigcup_{\gamma \in I_k} \sigma'_{\gamma i}) = \bigcup_{\gamma} \tau(\sigma'_{\gamma i}) = \bigcup_{\gamma} \tau(\bigcup_{\lambda = 1}^{\mu} \partial_\lambda \sigma_\gamma \Delta_{\lambda i}) = \bigcup_{\gamma} \partial_i \tau(\sigma_\gamma) =$$

$$= \partial_i \bigcup_{\gamma} \tau(\sigma_\gamma) = \partial_i \tau(\bigcup_{\gamma} \sigma_\gamma) = \partial_i \tau(\sigma''_k)$$

(we used Lemma 1 and the property that τ and ∂ "commute" with respect to $\cup$). Thus

$$\tau(\sigma''_{ki}) = \partial_i \tau(\sigma''_k)\,,$$

which is the same as writing

$$s_{ki} = \partial_i s_k\,.$$

Since the first rows of $\mathbf{\Gamma}$ and $\mathbf{\Gamma}_3$ are in correspondence by τ we see that the construction of $\mathbf{\Gamma}_3$ only simulates that of $\mathbf{\Gamma}$. Recognizing equivalences of entries (expressions in $L(\Xi)$) may introduce no principal error (this is guaranteed partly by Lemma 2, partly by the fact that overlooking some equivalences doesn't matter since we are proving only the equivalence of automata).

It remains to show the correctness of specifying the final states (step 4 of the procedure). But it follows immediately from the properties of $\varrho : \varrho(\tau(\alpha)) = \varrho(\alpha)$ and $\varrho(\alpha \cup \beta) = \varrho(\alpha) \cup \varrho(\beta)$, Q.E.D.

We have shown that the synthesis procedure $\alpha \to \mathscr{A}_\alpha \to \pi(\mathscr{A}_\alpha)$ if formally equivalent to the procedure $\alpha \to \tau(\alpha) \to \mathscr{A}_{\tau(\alpha)}$. As for the practical applications of these methods, it is difficult to present any general results. One may profitably use the former when the alphabet Ξ is considerably simpler then X and when numerous derivations are not desirable. The latter, however, more reliably guarantees the minimality of results.

5. DIRECT SYNTHESIS OF LOGICAL NET

In Section 4 we have introduced an auxiliary automaton $\mathscr{A}_\alpha$ over Ξ and we have shown that it is possible to progress from it to the proper abstract automaton over X by means of transformation π (Procedure 1). Now we shall use the auxiliary automaton $\mathscr{A}_\alpha$ for a direct synthesis of a logical net realizing $\mathscr{A}_{\tau(\alpha)}$. We shall enlarge somewhat the diagram from the proceding section:

$$\begin{tikzcd}
\alpha \in L(\Xi) \arrow[r, dashed, "\tau"] \arrow[d, dashed] & \tau(\alpha) \in L(X) \arrow[d, dashed] \\
\mathscr{A}_\alpha \arrow[r, dashed, "\pi", "\text{Proc. 1}"'] \arrow[d, "\text{Proc. 2}"'] & \mathscr{A}_{\tau(\alpha)} \arrow[d, dashed, "\boldsymbol{f},\, \boldsymbol{g}\text{-assignment}"] \\
\mathscr{N}_\alpha & \mathscr{A}^{(2)} \arrow[l, dashed, "\text{logical design}"]
\end{tikzcd}$$

Here $\mathscr{A}^{(2)}$ is some suitable binary automaton, $\mathscr{N}_\alpha$ is a logical net realizing $\mathscr{A}_{\tau(\alpha)}$.

We shall be concerned with the transition from $\mathscr{A}_\alpha$ to the corresponding logical net (solid arrow on the diagram). The procedure we shall describe is due, to a certain degree, to the intuitive view on the auxiliary automaton (see Remark on page 12) and it consists in associating some input variable (input terminal) with each symbol of Ξ and some internal variable (selected connection) with each state of Σ. In general case, however, when symbols of Ξ do not correspond to signals on input channels (comp. Example 2), a suitable *input decoder* must be supplied. Here we shall not be concerned with its construction.

The reader familiar with the paper of Yamada [10] will find close relationship between our nets and the so called disjunctively linear nets. This relationship will be studied more deeply later.

First we must concern ourselves with the connection of nets and automata from behavioral point of view. For the logical nets there is usually no state considered as initial and thus the binary automata describing them are noninitial. On the other hand we have defined abstract automata as initial (which is desirable from the behavioral point of view). A logical net may display the behavior of a given automaton only with respect to a specified starting state.

For our purposes it will be useful to introduce the so called initial net, provided with a special starting input. Assume that the net $\mathscr{N}$ remains in some specified *quiescent state* (e.g. all internal variables have value 0) independently on what is going on the input terminals. But a special *starting terminal* belongs to the net, and only at the moment when the signal 1 is applied to it, the net passes to the specified *initial state* $\boldsymbol{s}_1$ (which, however, may be identical with the quiescent state, in trivial case). From the subsequent instant of time the net is sensitive to input signals. In general case the net may sometimes return both to the quiescent and the initial state.

Definition 5. The logical net $\mathscr{N}$ with a starting terminal, having the described properties is called the *initial logical net.*

Remark. This approach is justified by the following fact: If the behavior of the net is to depend on certain finite sequences of input states, the net must necessarily obtain the information *at what time* these sequences begin (i.e. where the origin of discrete time is laid) and, moreover, it must be set to a difinite starting state.

Let $\mathcal{N}$ be an initial logical net wtih p input terminals and with the starting terminal which is set to 1 just at the time $t = 0$. Such net is then described by unique (up to the permutation of components) binary automaton $\mathcal{A}^{(2)}_{\mathcal{N}}$, which is initial.

Let $\mathcal{A}$ be an abstract automaton over an input alphabet X and let $\mathbf{f} : X \to \{0, 1\}^p$ be an input assignment of $\mathcal{A}$. We extend it by natural way to $\mathbf{f} : X^* \to (\{0, 1\}^p)^*$.

Definition 6. The net $\mathcal{N}$ is said to *realize the automaton* $\mathcal{A}$ *under the input assignment* $\mathbf{f}$ if

$$\forall u \in X^* , \quad \mathbf{f}(u) \in T(\mathcal{A}^{(2)}_{\mathcal{N}}) \quad \text{iff} \quad u \in T(\mathcal{A}) .$$

($T(\mathcal{A}^{(2)}_{\mathcal{N}})$ and $T(\mathcal{A})$ are sets of words recognized by $\mathcal{A}^{(2)}_{\mathcal{N}}$ and $\mathcal{A}$ respectively.)

Remark. In Definition 6 we say nothing about such sequences $\mathbf{u} \in (\{0, 1\}^p)^*$ that $\mathbf{f}^{-1}(\mathbf{u}) = \emptyset$. They are the so called "don't care" sequences and it is without importance for us, whether they belong to $T(\mathcal{A}^{(2)}_{\mathcal{N}})$ or not.

Immediately from Definition 6 we have

Lemma 3. *An initial logical net* $\mathcal{N}$ *realizing an automaton* $\mathcal{A}$ *also realizes every automaton equivalent to* $\mathcal{A}$ *(under the same input assignment).*

Definition 6 is, however, purely behavioral and gives no means for the decision whether the given net realizes the given automaton. Instead of deep investigation of this problem we present only a comparatively weak, but for our purposes quite sufficient result.

For this, let $\mathcal{A} = \langle X, S, s_1, \delta, F \rangle$ be an abstract automaton and let $\mathcal{N}$ be an initial logical net with a binary automaton $\mathcal{A}^{(2)}_{\mathcal{N}} = \langle \{0, 1\}^p, \{0, 1\}^q, \mathbf{s}_1, \boldsymbol{\delta}, \lambda \rangle$ describing it. Let $\mathbf{f} : X \to \{0, 1\}^p$ be an input assignment of $\mathcal{A}$.

Lemma 4. *The sufficient condition, under which the net* $\mathcal{N}$ *may realize the automaton* $\mathcal{A}$ *under the input assignment* $\mathbf{f}$ *is the existence of a state assignment* $\mathbf{g} : S \to \{0, 1\}^q$ *such that*

(1) $\mathbf{g}(s_1) = \mathbf{s}_1$,

(2) $\forall x \in X , \forall s \in S , \quad \boldsymbol{\delta}(\mathbf{g}(s), \mathbf{f}(x)) = \mathbf{g}(\delta(s, x))$,

(3) $\forall s \in S , \quad s \in F \quad \text{iff} \quad \lambda(\mathbf{g}(s)) = 1$.

Proof. Condition (2) may be rewritten as

$$\forall u \in X^* , \quad \forall s \in S , \quad \boldsymbol{\delta}(\mathbf{g}(s), \mathbf{f}(u)) = \mathbf{g}(\delta(s, u)) ,$$

especially for $s = s_1$ we use (1) to obtain

$$\forall u \in X^* , \quad \delta(\mathbf{s}_1, \mathbf{f}(u)) = \mathbf{g}(\delta(s_1, u)) .$$

Since $u \in T(\mathscr{A})$ iff $\delta(s_1, u) \in F$ and because of (3) we have $\delta(s_1, u) \in F$ iff $\lambda(\mathbf{g}(\delta(s_1, u))) = 1$ but $\lambda(\boldsymbol{\delta}(\mathbf{s}_1, \mathbf{f}(u))) = 1$ iff $\mathbf{f}(u) \in T(\mathscr{A}_{\mathscr{N}}^{(2)})$ and thus $\forall u \in X^*$, $\mathbf{f}(u) \in T(\mathscr{A}_{\mathscr{N}}^{(2)})$ iff $u \in T(\mathscr{A})$, Q.E.D.

In our cases the input assingment will always be given by the definition of Ξ. Let us define it more precisely:

Definition 7. Let X be an alphabet and $\Xi = \{\xi_1, \ldots, \xi_\mu\}$ some generalized alphabet to X, specified by the mapping

$$\varphi : \Xi \to \mathscr{P}(X) .$$

An assignment $\mathbf{f} : X \to \{0, 1\}^\mu$ is said to be a *natural input assignment* (with respect to Ξ) if

$$f_\lambda(x) = \begin{cases} 1 \text{ if } x \in \varphi(\xi_\lambda) \\ 0 \text{ otherwise} \end{cases}$$

where f_λ is the λ-th component of $\mathbf{f}$, $\lambda = 1, \ldots, \mu$.

The natural input assignment is unique up to order of symbols in Ξ and may be precisely determined by means of the mapping φ (see Definition 1) or by the transform matrix Δ (written over $\{0, 1\}$ instead of $\{\emptyset, \Lambda\}$).

Saying that a given net $\mathscr{N}$ "realizes an automaton $\mathscr{A}$" without specifying the input assignment, we shall always mean that $\mathscr{N}$ "realizes $\mathscr{A}$ under natural input assignment" supposing it is clear what specific Ξ is considered.

Now we shall describe the procedure leading from an auxiliary automaton $\mathscr{A}_\alpha$ over the generalized alphabet Ξ to some special logical net $\mathscr{N}_\alpha$ which, as we shall see later, realizes the proper automaton $\mathscr{A}_{\tau(\alpha)}$ over X.

Procedure 2. (We begin with an automaton $\mathscr{A}_\alpha$, given by its transition $\nu \times \mu$-table Γ_1 and by a set Φ of final states.)

It will be used altogether:

ν unit delays $\mathfrak{D}_1, \ldots, \mathfrak{D}_\nu$;

$\nu \times \mu$ two-input AND gates $\mathfrak{P}_{11}, \ldots, \mathfrak{P}_{\nu\mu}$;

$\nu + 1$ multiple-input OR gates $\mathfrak{S}_0, \mathfrak{S}_1, \ldots, \mathfrak{S}_\nu$.

The net $\mathscr{N}_\alpha$ will have μ input terminals denoted by the symbols $\xi_1, \ldots, \xi_\mu$, one starting terminal and one output terminal η.

1. Connect for each $\varkappa$ and λ ($\varkappa = 1, \ldots, \nu$; $\lambda = 1, \ldots, \mu$) one input of $\mathfrak{P}_{\varkappa\lambda}$ to the output of $\mathfrak{D}_\varkappa$ and the other input of $\mathfrak{P}_{\varkappa\lambda}$ to the terminal ξ_λ (It is convenient to arrange

the elements $\mathfrak{P}_{\varkappa\lambda}$ into a rectangular array metching the table $\mathbf{\Gamma}_1$ – compare Fig. 1 below.)

2. Connect for each γ $(\gamma = 1, \ldots, \nu)$ the individual inputs of $\mathfrak{S}_\gamma$ to outputs of all such elements $\mathfrak{P}_{\varkappa\lambda}$ for which the symbol σ_γ is in the $\varkappa$-th row and λ-th column of $\mathbf{\Gamma}_1$. Furthermore, connect one input of $\mathfrak{S}_1$ with the starting terminal.
3. Connect for each γ $(\gamma = 1, \ldots, \nu)$ the input of $\mathfrak{D}_\gamma$ to the output of $\mathfrak{S}_\gamma$.
4. Connect individual inputs of $\mathfrak{S}_0$ to outputs of all elements $\mathfrak{S}_\gamma$ for which $\sigma_\gamma \in \Phi$.
5. Connect the terminal η with the output of $\mathfrak{S}_0$. Stop.

(The result is the logical net $\mathcal{N}_\alpha$).

As already mentioned the net $\mathcal{N}_\alpha$ obtained by this procedure must be, in general case, provided by a suitable input decoder. This is a combinational network which has μ output terminals for $\xi_1, \ldots, \xi_\mu$. We suppose the input alphabet (and thus the input assignment) is already given and thus the construction of the decoder is not concerned in the procedure (it depends e.g. on the way how signals from environment are coming).

The construction of the net $\mathcal{N}_\alpha$ in Procedure 2 is apparent from the following example.

Example 4. Given an automaton $\mathcal{A}_\alpha$ by the transition table $\mathbf{\Gamma}_1$

	ξ_1	ξ_2	ξ_3
σ_1	σ_3	σ_2	σ_1
σ_2	σ_1	σ_2	σ_1
σ_3	σ_3	σ_2	σ_3

and by the final set $\Phi = \{\sigma_2, \sigma_3\}$. The corresponding net is shown in Fig. 1.

Remark. In particular cases it is possible to simplify considerably both the procedure and result by means of auxiliary rules, as for example:

(1) If some $\sigma_\gamma = \emptyset$ then all the elements $\mathfrak{S}_\gamma$, $\mathfrak{D}_\gamma$, $\mathfrak{P}_{\gamma\lambda}$ $(\lambda = 1, \ldots, \mu)$ as well as those elements $\mathfrak{P}_{\varkappa\lambda}$, for which the symbol $\emptyset$ is in the $\varkappa$-th row and λ-th column of $\mathbf{\Gamma}_1$, may be omitted.

(2) If some $\sigma_\gamma = \Lambda$ then the elements $\mathfrak{D}_\gamma$, $\mathfrak{P}_{\gamma\lambda}$ $(\lambda = 1, \ldots, \mu)$ may be omitted.

Now we shall prove the main theorem of this section:

Theorem 2. *Let $\alpha \in L(\Xi)$ and let $\mathcal{A}_\alpha$ be the corresponding auxiliary automaton over Ξ. Let $\mathcal{A}_{\tau(\alpha)}$ be the proper automaton over X.*

Then the logical net $\mathcal{N}_\alpha$ constructed from $\mathcal{A}_\alpha$ by the Procedure 2 *realizes the automaton $\mathcal{A}_{\tau(\alpha)}$ (under natural input assignment with respect to Ξ).*

Proof. Because of the equivalence of $\mathscr{A}_{\tau(\alpha)}$ and $\pi(\mathscr{A}_\alpha)$ (Theorem 1) it is sufficient to show that $\mathscr{N}_\alpha$ realizes $\pi(\mathscr{A}_\alpha)$ (see Lemma 3). We shall proceed as follows:

(I) We recapitulate the features of the automaton $\pi(\mathscr{A}_\alpha)$ and

(II) characterize the binary automaton $\mathscr{A}_{\mathscr{N}}^{(2)}$ that describes the net $\mathscr{N}_\alpha$; then

(III) we define some special state assignment $\boldsymbol{g}$ of $\pi(\mathscr{A}_\alpha)$ to $\mathscr{A}_{\mathscr{N}}^{(2)}$ and

(IV) conclude by showing that $\boldsymbol{g}$ is in concurrence with premises of Lemma 4.

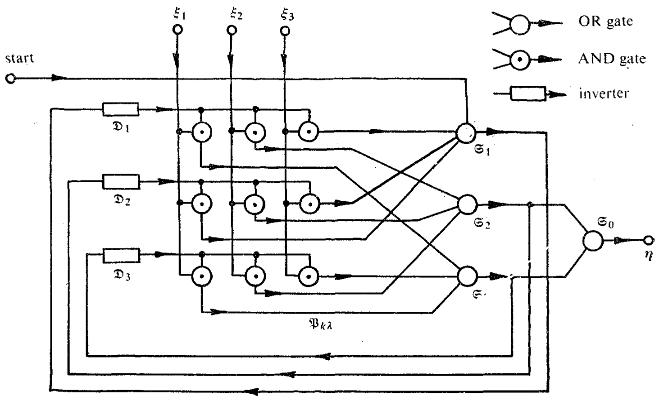

Fig. 1.

(I) Let $\mathscr{A}_\alpha = \langle \Xi, \Sigma, \sigma_1, \delta', \Phi \rangle$, where $\Xi = \{\xi_1, \ldots, \xi_\mu\}$, $\Sigma = \{\sigma_1, \ldots, \sigma_\nu\}$. Then $\pi(\mathscr{A}_\alpha)$ is defined as $\pi(\mathscr{A}_\alpha) = \langle X, S, s_1, \delta, F \rangle$ where $X = \{x_1, \ldots, x_m\}$ is related to Ξ by given injective mapping $\varphi : \Xi \to \mathscr{P}(X)$ (the symbols of Ξ denote some different subsets of X) and the set $S = \{s_1, \ldots, s_n\}$ have come to being in the course of Procedure 1: Each $s_k \in S$ may be written as the expression

$$s_k = \bigcup_{\gamma \in I_k} \sigma_\gamma \quad (I_k \subseteq \{1, \ldots, \nu\}, \sigma_\gamma \in \Sigma) .$$

In such case we shall say that σ_γ *occurs* in s_k. For s_1 we have especially $s_1 = \sigma_1$.

From Procedure 1 we may infer that some $\sigma_\varkappa$ occurs in $\delta(s, x)$ iff there exist σ_γ and ξ_λ such that

(a) σ_γ occurs in s;

(b) $x \in \varphi(\xi_\lambda)$;

(c) $\delta'(\sigma_\gamma, \xi_\lambda) = \sigma_\varkappa$.

Finally, $s \in F$ iff some $\sigma_\varkappa \in \Phi$ occurs in s.

(II) Now we specify the binary automaton $\mathscr{A}^{(2)}_{\mathscr{N}} = \langle \mathbf{X}, \mathbf{S}, \mathbf{s}_1, \boldsymbol{\delta}, \lambda \rangle$. Here $\mathbf{X} = \{0, 1\}^{\mu}$, $\mathbf{S} = \{0, 1\}^{\nu}$ and $\mathbf{s}_1 = (1, 0, \ldots, 0)$, because $\mathscr{N}_{\alpha}$ contains μ input terminals and ν delays and the starting terminal is connected just to the first delay (via the OR-gate $\mathfrak{S}_1$). For our purposes it will be useful to associate the internal states of the net with the values on *inputs* of delays (instead of its *outputs* as is usual).

It ramains to specify $\boldsymbol{\delta}$ and λ. From the internal behavior of $\mathscr{N}_{\alpha}$ it follows that the input of any delay $\mathfrak{D}_{\varkappa}$ ($\varkappa = 1, \ldots, \nu$) may have value 1 iff there exists at least one AND-gate $\mathfrak{P}_{\gamma\lambda}$ that

(a′) the output of the delay $\mathfrak{D}_{\gamma}$ has value 1 (i.e. its input had 1 one step before);

(b′) input terminal ξ_{λ} has value 1;

(c′) the output of $\mathfrak{P}_{\gamma\lambda}$ is connected to the input of the OR-gate $\mathfrak{S}_{\varkappa}$.

Thus, for $\mathscr{A}^{(2)}_{\mathscr{N}}$, if an input vector $\mathbf{x}$ and a state vector $\mathbf{s}$ are given, the state vector $\boldsymbol{\delta}(\mathbf{s}, \mathbf{x})$ is always uniquely determined by (a′), (b′) and (c′).

The output terminal η has the value 1 iff the output of some OR-gate $\mathfrak{S}_{\varkappa}$ connected to the OR-gate $\mathfrak{S}_0$ has value 1; thus we may define $\lambda(\mathbf{s}) = 1$ iff some $\varkappa \in \{1, \ldots, \nu\}$ exists, foı which $\sigma_{\varkappa} \in \Phi$ and the $\varkappa$-th component of $\mathbf{s}$ is 1.

(III) We define the state assignment $\mathbf{g} : S \to \{0, 1\}^{\nu}$ by the following way:

$$\mathbf{g}_{\varkappa}(s) = \begin{cases} 1 \text{ if } \sigma_{\varkappa} \text{ occurs in } s\,; \\ 0 \text{ otherwise} \end{cases}$$

where $\mathbf{g}_{\varkappa}$ is the $\varkappa$-th component of $\mathbf{g}$, $\varkappa = 1, \ldots, \nu$.

(IV) We prove that for $\pi(\mathscr{A}_{\alpha})$, $\mathscr{A}^{(2)}_{\mathscr{N}}$, natural input assignment $\mathbf{f}$ and just defined state assignment $\mathbf{g}$, the conditions (1), (2) and (3) of Lemma 4 are valid.

(1) From $s_1 = \sigma_1$ directly follows $\mathbf{g}(s_1) = (1, 0, \ldots, 0) = \mathbf{s}_1$.

(2) Let $s \in S$, $x \in X$. We are to prove that

$$\boldsymbol{\delta}(\mathbf{g}(s), \mathbf{f}(x)) = \mathbf{g}(\delta(s, x))$$

or componentwise

$$\boldsymbol{\delta}_{\varkappa}(\mathbf{g}(s), \mathbf{f}(x)) = 1 \quad \text{iff} \quad \mathbf{g}_{\varkappa}(\delta(s, x)) = 1$$

for all $\varkappa$. ($\boldsymbol{\delta}_{\varkappa}(\mathbf{s}, \mathbf{x})$ represents the $\varkappa$-th component of $\delta(\mathbf{s}, \mathbf{x})$).

Here $\mathbf{g}_{\varkappa}(\delta(s, x)) = 1$ iff $\sigma_{\varkappa}$ occurs in $\delta(s, x)$. This holds iff $\exists \sigma_{\gamma}\ \exists \xi_{\lambda}$ such that conditions (a) , (b) and (c) are fulfilled for $\sigma_{\varkappa}$. But (a) holds iff $\mathbf{g}_{\gamma}(s) = 1$ i.e. iff (a′) holds; (b) holds iff $\mathbf{f}_{\lambda}(x) = 1$ i.e. iff (b′) holds and the equivalence of (c) and (c′) follows from the construction of $\mathscr{N}_{\alpha}$ (Procedure 2, step 3). The conditions (a′), (b′) and (c′) are simultaneously valid for $\varkappa$ iff the input of $\mathfrak{D}_{\varkappa}$ in $\mathscr{N}_{\alpha}$ has value 1 and this is the same as $\boldsymbol{\delta}_{\varkappa}(\mathbf{g}(s), \mathbf{f}(x)) = 1$ in $\mathscr{A}^{(2)}_{\mathscr{N}}$.

(3) Let $s \in S$. Then $s \in F$ iff $\exists \varkappa [\sigma_\varkappa \in \Phi$ and $\sigma_\varkappa$ occurs in $s]$. This holds iff $\exists \varkappa [\sigma_\varkappa \in \Phi$ and $g_\varkappa(s) = 1]$ i.e. iff $\lambda(\mathbf{g}(s)) = 1$.

Now, by direct application of Lemma 4, the proof of Theorem 2 is completed. Q.E.D.

The complete synthesis leading from a given expression α over the generalized alphabet Ξ to the logical net $\mathscr{N}_\alpha$ realizing the desired automaton consists thus of two stages:

1. The synthesis of the auxiliary automaton $\mathscr{A}_\alpha$ over Ξ by means of derivatives in $L(\Xi)$;

2. Drawing the logical net $\mathscr{N}_\alpha$ which matches (in the sense of Procedure 2) the transition table of $\mathscr{A}_\alpha$.

This straightforward procedure of synthesis is comparatively very simple and easy. On the other hand, the resulting net is often not so economical as the net constructed by the usual method, which realizes an automaton by using an appropriate state-assignment procedure and logical design.

Our nets possess, however, some special features, which makes them very attractive. These features will be discussed in the following.

We already mentioned the paper [10] of Yamada concerning disjunctively linear logical nets (DL-logic nets). We do not intend to recapitulate here the Yamada's results and we only point out some similarities. Roughly speaking the nets yielded by the Procedure 2 belong to the class of DL-logic nets with the following two stipulations:

(1) The timing in our nets is slight different from that of Yamada, the starting signal being applied at the time immediately *preceding* the first significant signal on the input. This enables us to represent also star events, especially $\{\Lambda\}$. (In this point, our approach is due to Arden [1] and Brzozowski [2], whereas Yamada accepts that of Copi, Elgot and Wright [5]).

(2) DL-logic nets include a special input decoder (one-out-of-2^k code in the case of k input terminals). In our nets, this decoder is replaced by a quite *general decoder*; the most important case being the absence of any decoder at all. This generalization is of great theoretical as well as practical significance.

On the other hand, by the structure of the logic unit (in fact Procedure 2 yields only the logic unit of a net), our nets are special enough, and resemble those which Yamada calls the MM-logic nets (Sect. IV of [10]). The latter, however, differ in the property the *only one* delay element in the net is in state Y at any time, which corresponds to the state-assignment by one-out-of-n code. For completeness let us mention that similar nets are also presented in [8] (for asynchronous logic) and in [9].

From behavioral point of view, the most significant property of DL-logic nets is the so called disjunctively linear behavior. We shall present a slightly modified definition of it, to comply with our timing (accordingly it is the same as the definition of reccurent realization given by Brzozowski and Poage [4]).

For this, let $\mathcal{N}$ be an initial logical net and $\mathscr{A}^{(2)}$ its describing binary automaton.
Assume that the starting signal may be applied repeatedly at arbitrarily chosen times, and not only once at $t = 0$ as earlier.

Definition 8. The initial logical net $\mathcal{N}$ is said to have the *disjunctively linear behavior* if it has the following property: The output terminal of $\mathcal{N}$ has value 1 at time t iff there has been a signal 1 on the starting terminal at some time t' $(0 \leqq t' \leqq t)$ and the segment of the input sequence from $t' + 1$ to t (for $t' = t$ it is Λ) belongs to $T(\mathscr{A}^{(2)})$.

Now we may state

Theorem 3. *Any logical net $\mathcal{N}$ constructed by means of* Procedure 2 *and supplemented by arbitrary input decoder has disjunctively linear behavior.*

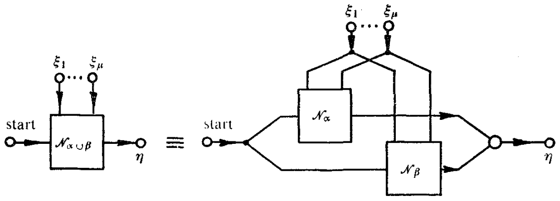

Fig. 2.

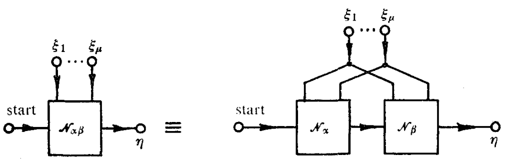

Fig. 3.

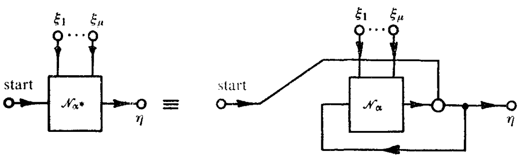

Fig. 4.

Proof will be omitted for it would entirely imitate that of Yamada (Property 2 in [10)]. The usage of the general decoder in our case has no influence on the proof.

It is the property of disjunctively linear behavior, what enables us to carry out compositions of nets in accordance whith basic Kleene's operations $\cup, \cdot, *$. (See [4] for the proof.) These compositions, used at first to illustrate the proof of Kleene's theorem (see [5]; for our modification see [1] and [2]), are best demonstrated by diagrams.

The net is generally represented by a square with its input terminals on the upper side, the starting terminal on the left and the output terminal on the right side.

The operation $\cup$ is represented by the parallel composition shown in Fig. 2. The operation $\cdot$ is represented by the cascade composition (Fig. 3). The operation $*$ is represented by the feedback loop (Fig. 4).

In practical design of logical nets it may be useful to combine these compositions with the synthesis described previously.

6. CONCLUSION

We have concerned ourselves with the application of the generalized alphabet Ξ and of the regular expressions over this alphabet to the description of the behavior of finite automata and to their synthesis. Our method has some advantages, e.g. that it enables us to synthetize the logical nets quickly and directly. It is obvious, that these advantages do not apply in all particular cases to the same degree. There are some reasons to recommend it, namely when dealing with multiple-input problems, where the behavior of automata depends more on the sequences of signals on individual inputs than on their instantaneous combinations; and still more if we are interested in simplifying more the method itself than its result.

Some restrictions accepted in this paper (e.g. the restriction to Moore automata with a single binary output) are not principal and the application of the method to more general cases should not prove to be too complicated. There is, however, one restriction which may not be omitted: the method of synthesis is not applicable to expressions of the extended language of regular expressions (with operations denoting the intersection and complementation of events).

Finally, another possible application of the generalized alphabet should be mentioned. There exist behavioral languages for finite automata, using formulas of symbolic logic (see e.g. [7] for details). These languages may be often useful because of their connection with the informal word description of the finite automata behavior. However, the synthesis procedure in logical languages is complicated and it is desirable to translate such formulas into another formalized language. The language of regular expressions over generalized alphabet is in this respect applicable.

Acknowledgment. The author wishes to thank to J. Š. Haškovec, CSc. for his careful guidance and interest in this work.

(*Received March 12th, 1968.*)

[1] D. N. Arden: Delayed Logic and Finite State Machines. In: Theory of Computing Machine Design, Univ. of Michigan Press, Ann Arbor 1960.

[2] J. A. Brzozowski: A. Survey of Regular Expressions and Their Applications. IRE Transactions *EC-11* (1962), 324—335.

[3] J. A. Brzozowski: Derivatives of Regular Expressions. Journal of the ACM *11* (1964), 481—494.

[4] J. A. Brzozowski, Poage: On the Construction of Sequential machines from Regular Expressions. IEEE Transactions *EC-12* (1963), 4, 402—403.

[5] I. Copi, C. Elgot, J. Wright: Realization of Events by Logical Nets. Journal of the ACM *5* (1958), 2, 181—196.

[6] М. А. Гаврилов: Синтез таблиц переходов методом обобщеных состояний входов. Автоматика и телемеханика (1967), 1, 89—99.

[7] I. M. Havel: Jazyky zápisu o zadání konečných automatů. Thesis. Praha 1966.

[8] P. R. Low, G. A. Maley: Flow Table Logic. Proceedings of the IRE *49* (1961), 221—228.

[9] K. Šiler: Některé způsoby logického návrhu kontrolní jednotky číslicového počítače. VÚMS, Praha 1965.

[10] H. Yamada: Disjunctively Linear Logic Nets. IRE Transactions *EC-11* (1962), 5, 623—639.

VÝTAH

Regulární výrazy nad zobecněnou abecedou a syntéza logických sítí

IVAN M. HAVEL

Je zaveden jazyk regulárních výrazů nad zobecněnou abecedou, jejíž symboly jsou interpretovány nikoliv jako vstupní stavy konečného automatu, nýbrž jako jisté množiny těchto stavů. Symboly této abecedy mohou např. odpovídat signálům na jednotlivých vstupních kanálech automatu s více vstupy.

Dále je ukázána metoda syntézy pomocí derivací výrazů, vedoucí od regulárního výrazu nad zobecněnou abecedou k tabulce přechodů abstraktního automatu nebo přímo ke schématu příslušné logické sítě.

Logické sítě vytvořené touto procedurou mají speciální charakter připomínající Yamadovy disjunktivně lineární sítě a umožňující kompozici automatů dle základních operací Kleeneho algebry.

Ing. Ivan M. Havel, Ústředí pro rozvoj automatizace a výpočetní techniky, Loretánské nám. 3, Praha 1.