Regular Expressions and State Graphs for Automata*

R. McNAUGHTON† AND H. YAMADA†

Summary—**Algorithms are presented for 1) converting a state graph describing the behavior of an automaton to a regular expression describing the behavior of the same automaton (section 2), and 2) for converting a regular expression into a state graph (sections 3 and 4). These algorithms are justified by theorems, and examples are given. The first section contains a brief introduction to state graphs and the regular-expression language.**

I. Terminology, Notation, and Fundamental Theorems

In this paper a state graph for a one-input, one-output automaton consists of a number of circles connected with arrows, each arrow being labeled 0 or 1, or both. Each circle shall have exactly one arrow leaving it labeled 0, and exactly one labeled 1, leading either back to the same circle or to another circle. Some circles (double circles) have within them a slightly smaller concentric circle while others (single circles) do not. The graph has a single unlabeled arrow pointing to one of the circles, but not leading *from* any circle. The single circles represent states, transitions into which yield an output 0; the double circles represent states, transitions into which yield an output of 1. The unlabeled arrow points to the initial state. These state graphs, like those of Moore,[1] have a single output associated with each state. Unlike the treatment in his work, however, the output of an automata in this paper occurs during the transition into a state, not when the automaton is in the state. Thus, the output of the initial state is not necessarily the first output of the output sequence. For example, if the input sequence 101001111 is applied to automaton described by the graph of Fig. 1, then the output sequence is 001110101. This type of state graph makes for convenience in the algorithms of the later sections.

Regular expressions are introduced which denote classes of sequences of zeros and ones, and which describe the behavior of automata. A sequence of zeros and ones is thought of as an input sequence of an automaton. The manner in which a regular expression describes the behavior of an automaton is described below. Before that, a precise definition of "regular expression" is given, followed by a precise account of how a regular expression denotes a class of sequences of zeros and ones.

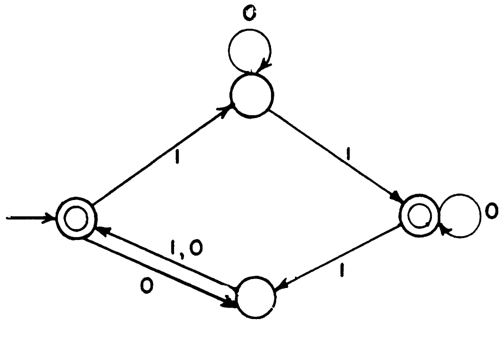

Fig. 1.

Regular Expressions

The regular-expression language (adapted from Kleene[2] and Copi, Elgot, and Wright,[3] has the symbols zero (0), one (1), star (*), dot (·), cup (∪), cap (∩), tilde (∼), (upper-case) lambda (Λ), (lower-case) phi (ϕ), and parentheses. Zero and one are digits. Digits, lambda, and phi are term signs. Star, dot, cup, cap, and tilde are the operator signs. Parentheses are used for grouping.

"Regular expression" is defined recursively as follows:

1) A string consisting of a single zero, a single one, a single lambda, or a single phi is a regular expression.

2) If $\beta_1, \cdots, \beta_n$ are regular expressions then so are $(\beta_1)^*$, $(\beta_1)\cdot(\beta_2)\cdot \cdots \cdot(\beta_n)$, $(\beta_1)\cup(\beta_2)\cup \cdots \cup(\beta_n)$, $(\beta_1) \cap \cdots \cap(\beta_n)$, and $\sim(\beta_1)$.

3) No string is a regular expression unless its being so follows from 1) and 2).

In practice the dot will be omitted completely. For convenience, parentheses will sometimes be omitted, in a systematic fashion, by specifying the order of strength of the various operator signs. Cup and cap have the greatest strength and are equal in strength; next is the dot; finally the tilde and star have the least strength. Thus $0\cup 10$ is short for $0\cup 1\cdot 0$. In the latter the cup has greater strength than the dot, so that the terms joined by the dot are grouped together before terms are grouped by the cup. Thus the fully unabbreviated form is $(0)\cup[(1)\cdot(0)]$. Similarly, $0\cup\sim 0$, $0\cap 0^*$, 00^* and ~ 00 are to be interpreted, respectively as $(0)\cup[\sim(0)]$, $(0)\cap[(0)^*]$, $(0)\cdot[(0^*)]$ and $[\sim(0)]\cdot(0)$. No parentheses in $(0\cup 1)(00)^*$ may be omitted.

* Manuscript received by the PGEC, August 5, 1959; revised manuscript received, December 14, 1959. This work was carried out under U. S. Air Force Contract AF 33(616)-5886, placed by Wright Air Development Center.

† Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia.

[1] E. F. Moore, "Gedanken experiments on sequential machines," in "Automata Studies," C. E. Shannon and J. McCarthy, Eds., Princeton, N. J., pp. 1–29–156; 1956.

[2] S. C. Kleene, "Realization of events in nerve nets and finite automata," in "Automata Studies," C. E. Shannon and J. McCarthy, Eds., Princeton, N. J., pp. 3–42; 1956.

[3] I. M. Copi, C. L. Elgot, and J. B. Wright, "Realization of events by logical nets," *J. Assoc. Comp. Mach.*, vol. 5, pp. 181–196; April, 1958.

Sequences

Regular expressions are used to denote sequences of zeros and ones. The *length* of a sequence is the number of terms (zeros and ones) in it. If $\sigma_1, \cdots, \sigma_n$ are finite sequences of zeros and ones whose lengths are $l_1, \cdots, l_n$, respectively, then the *concatenation* of $\sigma_1, \cdots, \sigma_n$ is the sequence σ of length $l_1 + \cdots + l_n$, such that, for each i, $1 \leqq i \leqq n$, the $(l_1 + \cdots + l_{i-1} + 1)$th through the $(l_1 + \cdots + l_i)$th terms are those of σ_i in order. Note that the concatenation of σ_1, σ_2 is different from the concatenation of σ_2, σ_1. If σ_1, or σ_2, is the null sequence (whose length is zero), then the concatenation of σ_1, σ_2 is σ_2, or σ_1, respectively.

A regular expression denotes both sequences and classes of sequences. Since a regular expression denotes only a single class of sequences and denotes a sequence if, and only if, the sequence is a member of that class, no confusion will result. The recursive characterization of denotation is as follows:

1) If a regular expression denotes a class of sequences, then it denotes any sequence in that class.

2) The regular expression 0 and the regular expression 1 denote, respectively, the unit class of the sequence of a single zero and the unit class of the sequence of a single one. (A unit class of an entity is the class whose only member is that entity.)

3) The regular expression ϕ denotes the unit class of the null sequence, *i.e.*, the sequence of zero length.

4) The regular expression Λ denotes the null class of sequences.

5) The regular expression β^* denotes the smallest class of sequences which contains the null sequence and contains, for any σ_1 denoted by β^* and σ_2 denoted by β, the concatenation of σ_1, σ_2. In other words, β^* denotes the class of all sequences σ such that, for some non-negative integer n (zero included!), σ is the concatenation of n sequences (not necessarily distinct and not necessarily identical), each of which is denoted by β.

6) The regular expression $\beta_1 \cup \beta_2 \cup \cdots \cup \beta_n$ denotes the union of the classes denoted by $\beta_1, \beta_2, \cdots, \beta_n$.

7) The regular expression $\beta_1 \cap \beta_2 \cap \cdots \cap \beta_n$ denotes their intersection.

8) The regular expression $\beta_1 \beta_2 \cdots \beta_n$ denotes the class of all σ's such that there exist sequences $\sigma_1, \cdots, \sigma_n$, where, for each i, σ_i is denoted by β_i, and σ is the concatenation of $\sigma_1, \cdots, \sigma_n$.

9) The regular expression $\sim\beta$ denotes the complement of the class denoted by β with respect to the class of all finite sequences of zero and ones.

10) A regular expression does not denote anything unless its doing so follows from 1) through 9).

Thus, for example, $00 \cup 11$ denotes the class whose members are just the two sequences 00 and 11. And $(00 \cup 11)^*$ denotes the infinite class whose members are the empty sequence, 00, 11, 0000, 0011, 1100, 1111, 000000, 000011, etc.

Note that lambda is used here to denote the empty class of sequences and not the empty sequence as it sometimes does in the literature in information theory. That the empty sequence and the empty class of sequences must be distinguished is apparent because of the different ways they occur in the equations at the end of this section.

Two regular expressions α and β are equal $(\alpha = \beta)$ if they denote the same class.

Automata

A regular expression α describes (the behavior of) a one-input, one-output automaton when, for every $t \geqq 1$, the output at time t is 1, if and only if the input sequence from time 1 through time t inclusive is denoted by α. Note that, according to this point of view, an automaton cannot respond to the null sequence of inputs. This results from our characterization of state graphs, in which the initial state does not necessarily yield an output. As a consequence, we have an important principle: a regular expression α describes the behavior of an automaton if and only if $\phi \cup \alpha$ describes its behavior.

Although only automata with one binary input and one binary output are considered, the results of this paper can be applied to automata with more than two input states and more than two output states. As far as state graphs are concerned, this fact is apparent from the literature. A regular expression can describe the behavior of an automaton with n input states if it uses the symbols $2, 3, \cdots, n-1$ as well as 0 and 1 as digits. If the automaton has more than two output states, however, the matter is not that easy. In this case, the only method seems to be to assume that there are b binary outputs, and to have a separate regular expression describing each. The regular-expression language seems most suitable for automata with only one binary output.

As an example, consider the modulo-two one's counter, whose state graph is shown in Fig. 2; its output is 1 at time t if, and only if, the input is 1 at time t and there have been an even number of 1's up to and including time t in the input sequence. A regular expression describing this automaton is $(0^*10^*1)^*$.

It will follow from the results of sections II, III, and IV that every regular expression describes a finite automaton and that every finite automaton is described by some regular expression. It also follows that regular expressions that contain no tilde, cap, lambda, or phi, are sufficient to describe all automata, except the trivial automaton described by Λ, which produces a zero output at all times regardless of the input sequence. This result and its converse appeared in Kleene's article,[2] and is further clarified by Copi, Elgot, and Wright.[3] The extension of this result to all regular expressions is a fairly easy matter, and requires far less effort than the con-

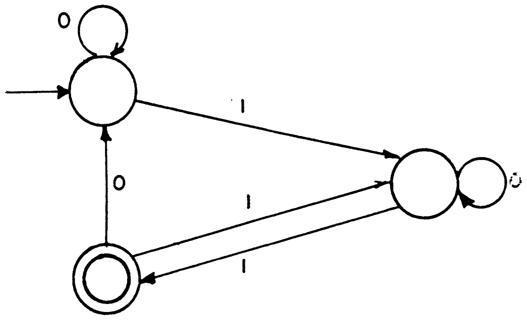

Fig. 2.

tents of sections II, III, and IV. (The verification of this fact we leave to the reader.) The purpose of those sections is the presentation of efficient ways of converting from regular-expression description of automata to state-graph description, and vice versa.

A Specification Language for Automata

The process of logic design is one that starts with a specification of what a logic circuit must do in the way of characterizing how the circuit responds with outputs to inputs. This specification must be precise, in order to produce by the process of logic design a result that is wanted.

We are of the opinion that the regular-expression language as presented here is a good language for such specifications. It has three virtues. First, in contrast to the state-graph language, its descriptions can be conveniently written out in a line from left to right. Second, it is precise and formal, as opposed, for example, to natural languages such as English. Third, there is indication that one who is trained in using the language can readily, that is, without computation or excessive reflection, express ideas in this language. This last virtue is an important one. The regular expression language without the tilde, cap, phi, and lambda (which was the language developed by Kleene[2] and Copi, Elgot, and Wright[3]) does not have this virtue, or has it to a lesser degree.

As an example, consider the problem of designing an automaton that has an output 1 at time t if and only if either there have never been three consecutive zeros in the input sequence up to and including time t, or there have been three consecutive ones in the input sequence since the last three consecutive zeros. A regular expression specifying the automaton can readily be determined. Noting that $(0\cup 1)^*$ denotes the class of all finite sequences (of zeros and ones), and that $(0\cup 1)^*000$ $(0\cup 1)^*$ denotes the class of all finite sequences having three consecutive zeros somewhere, the desired regular expression is

$$\sim[(0\cup 1)^*000(0\cup 1)^*]\cup$$
$$(0\cup 1)^*111\sim[(0\cup 1)^*000(0\cup 1)^*].$$

One can express this idea without the tilde, noting after some reflection that $(1\cup 01\cup 001)^*(\phi\cup 0\cup 00)$ denotes the class of all finite sequences that do not have three consecutive zeros anywhere. But this is not what one thinks when one says, "a sequence without three consecutive zeros anywhere." In a certain sense $\sim[0\cup 1)^*000(0\cup 1)^*]$ is what one thinks when one utters that English phrase.

If we are justified in believing that this regular-expression language is a good specification language for automata, then the algorithm of sections III and IV of this paper are important. They provide a way of obtaining a state graph from any regular expression; from the state graph the automaton can be designed, depending on the physical means at the designers' disposal. In Copi, *et al.*,[3] a method is given (which is a modification of one appearing in Kleene[2]) for constructing a net consisting of "and" gates, "or" gates, inverters, and delay elements from a regular espression not containing tilde, cap, lambda, and phi. (A minor adjustment must be made since, in their terminology, α^* denotes the result of concatenating members of α *one* or more times, instead of *zero* or more times as used here.) There seems to be no way of extending that algorithm to regular expressions containing tilde and cap.

Some Regular-Expression Equations

The following can be verified by using the characterization of "denotation" and other principles presented in this section:

$$\alpha\beta\cup\alpha\gamma = \alpha(\beta\cup\gamma),$$
$$\beta\alpha\cup\gamma\alpha = (\beta\cup\gamma)\alpha,$$
$$(\alpha\cup\beta)^* = (\alpha^*\cup\beta^*)^* = (\alpha^*\beta^*)^*,$$
$$\phi\alpha = \alpha\phi = \alpha,$$
$$\phi^* = \phi,$$
$$\phi\cup\alpha^* = \alpha^*.$$

Any rule of Boolean Algebra holds for cup, cap, tilde, and lambda as union, intersection, complementation, and null set, respectively.

$$\Lambda^* = \phi,$$
$$\alpha\Lambda = \Lambda\alpha = \Lambda,$$
$$\sim((0\cup 1)^*) = \Lambda,$$
$$\sim((0\cup 1)(0\cup 1)^*) = \phi,$$
$$\beta_1\cdots\beta_i\beta_{i+1}\cdots\beta_n = (\beta_1\cdots\beta_i)\beta_{i+1}\cdots\beta_n$$
$$= \beta_1\cdots\beta_i(\beta_{i+1}\cdots\beta_n), \quad 1\leqq i\leqq n-1.$$

Note that the following pairs of regular expressions are not equal, as the accompanying counterexamples show.

$$\alpha\beta \cap \alpha\gamma, \qquad \alpha(\beta \cap \gamma). \tag{1}$$

Example: $\alpha = 0 \cup 00,$

$\beta = 0,$

$\gamma = 00.$

$$(\alpha \cup \beta)^*, \qquad \alpha^* \cup \beta^*. \tag{2}$$

Example: $\alpha = 0,$

$\beta = 1.$

$$(\alpha \cap \beta)^*, \qquad \alpha^* \cap \beta^*. \tag{3}$$

Example: $\alpha = 00,$

$\beta = 000.$

II. An Algorithm for Constructing a Regular Expression from a State Graph

Let G be a state graph with n states $S_1, \cdots, S_n$. (S_1 is not necessarily the initial state.) Let $\alpha_{ij}{}^k$ be defined for all i, j, k such that $n \geqq k \geqq 0$, and $n \geqq i \geqq 1$, and $n \geqq j \geqq 1$, by induction on k as follows:

$\alpha_{i,j}{}^0 = 0 \cup 1$, if input 1 and input 0 both take G from S_i to S_j,

$= 0$, if input 0 but not input 1 takes G from S_i to S_j,

$= 1$, if input 1 but not input 0 takes G from S_i to S_j, and

$= \Lambda$, if neither 0 nor 1 takes G from S_i to S_j.

For $n \geqq k \geqq 1$, assume that $\alpha_{xy}{}^{k-1}$ has been defined for all x and y. Then

$$\alpha_{ij}{}^k = \alpha_{ij}{}^{k-1} \cup \alpha_{ik}{}^{k-1}(\alpha_{kk}{}^{k-1})^* \alpha_{kj}{}^{k-1}.$$

Theorem 2.1. $\alpha_{ij}{}^k$ denotes the class of all input sequences that take the state graph from S_i to S_j without going through S_x for any $x > k$. (Note the meaning of the word "through" here. If $i = 4$, $j = 10$, and the state graph goes from S_4 to S_{10} by means of the transition $S_4S_5S_4S_6S_8S_{10}$, then it goes through S_4 but not through S_{10}.)

The proof is by induction on k, following the recursive definition of $\alpha_{ij}{}^k$. For $k = 0$ it must be proved that $\alpha_{ij}{}^0$ denotes the class of sequences taking the graph from S_i to S_j without passing through any states at all. There are only two possible such sequences: the one-term sequence 0 and the one-term sequence 1. Hence, $\alpha_{ij}{}^0$ as defined, satisfies the condition of the theorem.

Assume now that $\alpha_{ij}{}^{k-1}$ satisfies the condition for all i and j. Each sequence taking the automaton of the state graph from S_i to S_j without causing it to pass through any S_x for $x > k$ could cause it to pass through S_k any number of times.

Case I. It causes it to pass through S_k zero times. Then the sequence is denoted by $\alpha_{ij}{}^{k-1}$, by inductive hypothesis; hence it is denoted by $\alpha_{ij}{}^k$ as constructed.

Case II. It causes it to pass through S_k exactly once. Let QR be the sequence, with Q the portion that leads from S_i to S_k, and R the portion that leads from S_k to S_j. Then Q is denoted by $\alpha_{ik}{}^{k-1}$, and R by $\alpha_{kj}{}^{k-1}$, by inductive hypothesis. Hence, by the meaning of the star, operator QR is denoted by $\alpha_{ik}{}^{k-1}(\alpha_{kk}{}^{k-1})^*\alpha_{kj}{}^{k-1}$ and, hence, by $\alpha_{ij}{}^k$ as constructed.

Case III. It causes it to pass through S_k more than once. Let QPR be the sequence, where Q is the portion taking it from S_i to S_k the first time, R is the portion taking it from S_k the last time to S_j, and P is the intervening portion (which takes it from S_k to S_k). Q is denoted by $\alpha_{ik}{}^{k-1}$, R by $\alpha_{kj}{}^{k-1}$ and P by $(\alpha_{kk}{}^{k-1})^*$. Hence QPR is denoted by $\alpha_{ij}{}^k$ as constructed.

This proves that $\alpha_{ij}{}^k$ denotes all the sequences satisfying the construction. Using the inductive hypothesis, one can easily see that it satisfies only such sequences.

Theorem 2.2. If an automaton is described by a state graph with states $S_1, \cdots, S_n$ whose initial state is S_i and whose states of output 1 are $S_{u_1}, \cdots, S_{u_m}$, then the automaton is described by the regular expression

$$\alpha_{iu_1}{}^n \cup \cdots \cup \alpha_{iu_m}{}^n.$$

This theorem is a direct consequence of theorem 2.1. Note that if the state graph of an automaton has no states of output 1 then Λ describes the automaton. Otherwise a regular expression is given by theorem 2.2. Thus, every automaton is described by some regular expression.

Although the method of construction is somewhat complicated, we have found no more direct general method for converting a state graph to a regular expression. Part of the problem is that it is rather difficult to get a regular expression denoting the input sequences of all paths from S_i to S_j, since these may involve many loops.

To obtain a regular expression from a state graph, one must first assign an order to the states. Different orderings will, in general, result in different regular expressions. One should attempt to order the states so that the simplest regular expression results. One strategic consideration is that those states bearing the most traffic (those with the largest number of arrows pointing to them) should come later in the ordering.

If we count the number of digits in a regular expression as its size, then the maximum size of $\alpha_{ij}{}^k$ is $2 \cdot 4^k$. A reduced n-state graph has a maximum of $n-1$ states with output 1. (If all n states had output 1 then the graph would reduce to a one-state graph.) Hence, the maximum size of a regular expression constructed from an n-state graph is $(n-1)2 \cdot 4^n$. Actually it will usually be much less than this, because simplifications can often be made along the way, as when $i=k$ or $j=k$. Thus α_{ij}^{i} is by construction

$$\alpha_{ij}^{i-1} \cup \alpha_{ii}^{i-1}(\alpha_{ii}^{i-1})^{*}\alpha_{ij}^{i-1},$$

which reduces to

$$(\alpha_{ii}^{i-1})^{*}\alpha_{ij}^{i-1}.$$

Likewise α_{ij}^{j} reduces to

$$\alpha_{ij}^{j-1}(\alpha_{jj}^{j-1})^{*}.$$

However, the size of the regular expression, though generally much less than $(n-1)2\cdot 4^{n}$, is still rather large. We are of the opinion that the simplest regular expression describing an automaton is usually quite large in comparison to the simplest state graph.

As an example, consider the state graph of Fig. 3. A regular expression for it is α_{21}^{3}. We find that

$$\alpha_{21}^{2} = (10)^{*}1,$$

$$\alpha_{23}^{2} = (10)^{*}(11 \cup 0),$$

$$\alpha_{33}^{2} = 0 \cup 1(10)^{*}(11 \cup 0),$$

$$\alpha_{31}^{2} = 1(10)^{*}1.$$

(It is an easy matter to obtain these by inspection from the state graph. Alternatively they can be obtained by means of the algorithm, making use of simplifications based on laws developed in section I.) Thus,

$$\alpha_{21}^{3} = (10)^{*}1 \cup (10)^{*}(11 \cup 0)[0 \cup 1(10)^{*}(11 \cup 0)]^{*}1(10)^{*}1.$$

On the other hand, if the states are labeled as in Fig. 4, the desired regular expression is α_{31}^{3}. We find that

$$\alpha_{33}^{2} = 10 \cup (0 \cup 11)0^{*}1,$$

$$\alpha_{31}^{2} = 1,$$

and therefore,

$$\alpha_{31}^{3} = [10 \cup (0 \cup 11)0^{*}1]^{*}1.$$

Although this regular expression and the one obtained previously describe the same automaton and are equivalent, this equivalence is far from obvious.

III. An Algorithm for Constructing a State Graph of an Automaton from a Restricted Regular Expression

A restricted regular expression is one which has no cap, tilde, phi, or lambda. Associate with each occurrence of a digit in a restricted regular expression a *position*, which can best be regarded as being directly to the right of the digit. Then in generating a sequence denoted by a regular expression one can think of going from position to position, through the digit of the latter position. For example, consider the restricted regular expression $((0\cdot 0 \cup 1\cdot 1)^{*}0\cdot 1)^{*}$. There are six positions in this expression. From left to right let these be 0_1 (the position to the right of the first zero), 0_2, 1_1, 1_2, 0_3, and 1_3. An alternative formulation of the notion of a sequence denoted by a restricted regular expression can be given in terms of these positions.

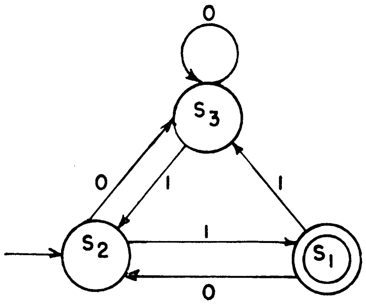

Fig. 3.

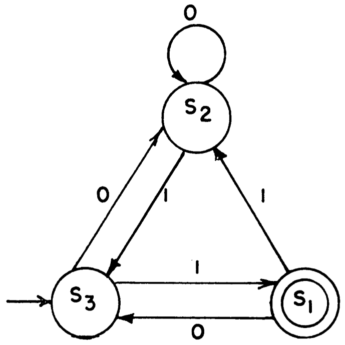

Fig. 4.

In the definitions that follow, restricted regular expressions that denote the empty sequence are distinguished from those that do not. The following recursive characterization of this property for the general class of regular expressions is given, because of its usefulness in the next section as well as in this section. 1) ϕ denotes the empty sequence, but Λ, 0, 1 do not. 2) For every α, α^{*} denotes the empty sequence. 3) For every α, $\sim\alpha$ denotes the empty sequence if and only if α does not. 4) $\alpha_1 \cup \cdots \cup \alpha_n$ denotes the empty sequence if and only if at least one α_i does. 5) $\alpha_1\alpha_2 \cdots \alpha_n$ denotes the empty sequence if and only if every α_i does. 6) $\alpha_1 \cap \alpha_2 \cap \cdots \cap \alpha_n$ denotes the empty sequence if and only if every α_i does.

For the purposes of this section, we do not have to refer to 3) and 6) of this definition. It is worthwhile to note that no restricted regular expression without a star denotes the empty sequence.

Let α be a well-formed part of a restricted regular expression ρ. Then "terminal position of α" is defined recursively as follows: 1) If α has only a single position then that position is a terminal position of α. 2) If α is $\beta_1 \cup \beta_2 \cup \cdots \cup \beta_n$ then any terminal position of any of the β's is a terminal position of α. 3) A terminal position of β is a terminal position of β^*. 4) If α is $\beta_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n$ and if for every x, $n \geqq x > j$, β_x denotes the empty sequence, then a terminal position of β_j is a terminal position of α; in particular, a terminal position of β_n is a terminal position of α. 5) No position is a terminal position of α unless its being so follows from 1), 2), 3), and 4).

"Initial position of α" is defined recursively as follows: 1) If α has only a single position then that position is an initial position of α. 2) If α is $\beta_1 \cup \cdots \cup \beta_n$ then an initial position of any of the β's is an initial position of α. 3) An initial position of β is an initial position of β^*. 4) If α is $\beta_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n$ and for all x, $1 \leqq x < j$, β_x denotes the empty sequence, then an initial position of β_j is an initial position of α. (Thus, in any case, an initial position of β_1 is an initial position of α.) 5) Nothing is an initial position of α unless its being so follows from 1), 2), 3), and 4).

It can be proved that the left-most position of α is always an initial position of α, and likewise that the right-most position of α is a terminal position of α, although these are not always the only initial and terminal positions. For example, in $(0_1 \cup 1_1)^* 0_2 0_3$, 0_2 and 1_1 as well as 0_1 are initial positions, although 0_3 is the only terminal position.

A transition of a restricted regular expression ρ is an ordered pair of positions $\langle P_1 P_2 \rangle$ such that, either P_1 is a terminal position of α, P_2 is an initial position of β, and $\gamma_1 \cdot \cdots \cdot \gamma_m \cdot \alpha \cdot \delta_1 \cdot \cdots \cdot \delta_n \cdot \beta \cdot \eta_1 \cdot \cdots \cdot \eta_q$ ($m \geqq 0$, $n \geqq 0$, $q \geqq 0$) is a well-formed part of ρ where each of the δ's denotes the empty sequence, or P_1 is a terminal position of α^* and P_2 is an initial position of α^*, where α^* is a well-formed part of ρ.

An allowable sequence of positions of ρ is a finite sequence of positions $P_1, \cdots, P_n$ of ρ such that P_1 is an initial position of ρ, P_n is a terminal position of ρ and, for each $i < n$, $\langle P_i P_{i+1} \rangle$ is a transition of ρ.

Theorem 3.1. A finite sequence S of 0's and 1's is denoted by a restricted regular expression ρ if and only if there exists an allowable sequence R of positions of ρ such that S results from R by replacing each position by the digit of that position.

This theorem follows from the above definitions and the definition of denotation given in section I. (A rigorous proof would be by induction, following the recursive definition of "regular expression," restricted. We trust that the construction of this proof would be an easy task for the interested reader.)

If a restricted regular expression ρ containing p positions describes the behavior of an automaton, a state graph containing 2^p+1 states can be constructed by the method given below. It will be proved that this state graph is a state graph for the given automaton. (In practice, usually a far simpler state graph will suffice. It will turn out that many of the 2^p+1 states will not be used.)

Construction method. Let ρ have p positions. Then let there be an initial state, and, in addition, one state for each set of positions (thus giving 2^p+1 states). An input of 0 applied to a state corresponding to a set S of positions will lead to the state corresponding to the set S' of just those positions P such that P's digit is 0 and there is a transition from at least one position of S to P. The same statement applies for an input of 1. If there is no such position P then the preceding sentence implies that it leads to the state corresponding to the null set of positions; also implied is that an input applied to the state corresponding to the null set results again in that state. An input of 0 applied to the initial state results in the set of all positions P which are initial positions of ρ and whose digit is 0. The same statement applies for an input of 1. A state has an output of 1 if and only if there is at least one position in the corresponding set of positions which is a terminal position of ρ.

In discussing this construction method we shall verbally identify a state of the resulting state graph with the corresponding set of positions.

Theorem 3.2. The state graph G obtained from ρ by the above construction method describes the behavior of the same automaton as ρ.

By theorem 3.1, it is sufficient to prove, for any sequence of inputs $i_1, \cdots, i_n$ that takes the state graph from the initial state through a series of states $S_1, \cdots, S_n$, that S_n is a state of output 1 if and only if there exists an allowable sequence of positions $P_1, \cdots, P_n$ such that for each j, $1 \leqq j \leqq n$, i_j is the digit of P_j.

Lemma. For each S_j of the sequence $S_1, \cdots, S_n$, all positions of S_j have the same digit, namely i_j.

The proof is immediate from the construction. (This lemma shows that many of the 2^p+1 states are dispensable in the state graph, namely, those states some of whose positions have the digit 0 and others have 1. Usually, many states other than those shown to be dispensable by this lemma are also dispensable.)

To proceed with the proof of theorem 3.2, suppose first that S_n is a state of output 1. Then, by construction, there exists a position P_n in S_n such that P_n is terminal in ρ. There must exist a position P_{n-1} of S_{n-1} such that $\langle P_{n-1} P_n \rangle$ is a transition of ρ; otherwise P_n would not be in S_n, by construction. Likewise there must be a P_{n-2} in S_{n-2} such that $\langle P_{n-2} P_{n-1} \rangle$ is a transition of ρ. And so on, back. The resulting sequence of positions $P_1, \cdots, P_n$ is an allowable sequence of positions of ρ, since P_1 is an initial position of ρ (for, by construction, S_1 is the set of initial positions of ρ having the digit i_1), for every $j < n$ $\langle P_j P_{j+1} \rangle$ is a transition of ρ, and P_n is a terminal position of ρ. Furthermore, by the lemma, for each j, i_j is the digit of P_j. This concludes one direction of the proof of theorem 3.2.

To prove theorem 3.2 the other way, suppose that there exists an allowable sequence of positions $P_1, \cdots, P_n$ such that, for each $j \leqq n$, i_j is the digit of P_j. By construction, S_1 must be the set of initial positions with i_1 as digit, including P_1. S_2 must have P_2 as a member, since $\langle P_1P_2\rangle$ is a transition of ρ. Likewise S_3 must have P_3 as a member, and so on. Thus S_n must have P_n as a member and therefore must be a state of output 1. This completes the proof of theorem 3.2.

We close this section with a practical method of obtaining the state graph from a restricted regular expression. Far fewer states than 2^p+1 are usually needed, so it is not expedient to assume this many states at the beginning. Rather the method constructs the state graph state by state, adding states only when they are needed. The method will be explained by means of a short example.

Let ρ be $1(00\cup 01)^*0$. There are six positions which are in order 1_1, 0_1, 0_2, 0_3, 1_2, 0_4. Rewriting ρ with positions for digits we get $1_1(0_10_2\cup 0_31_2)^*0_4$. 1_1 is the only initial position of ρ. 0_4 is the only terminal position of ρ. The transitions are

$$\begin{array}{lll}
\langle 1_10_1\rangle, & \langle 1_10_3\rangle, & \langle 1_10_4\rangle \\
\langle 0_10_2\rangle & & \\
\langle 0_31_2\rangle & & \\
\langle 0_20_1\rangle, & \langle 0_20_3\rangle, & \langle 0_20_4\rangle \\
\langle 1_20_1\rangle, & \langle 1_20_3\rangle, & \langle 1_20_4\rangle.
\end{array}$$

In constructing the state graph we first assume the initial state. We then ask, what state, *i.e.*, set of positions, do we get to by an initial input of 1 and what state do we get to by an initial input of 0? An input of 1 applied to the initial state I lands us in state $\{1_1\}$, whose only position is 1_1. An input of 0 applied to I lands us in the state Λ, corresponding to the null set of positions. Once in this state the automaton can never get out of it, so a descriptive name for it is "the dead state." An input of 0 applied to $\{1_1\}$ results in the state $\{0_1, 0_3, 0_4\}$; an input of 1 results in Λ. An input of 0 applied to $\{0_1, 0_3, 0_4\}$ results in the state $\{0_2\}$; an input of 1 results in the state $\{1_2\}$. An input of 1 applied to $\{0_2\}$ results in Λ; an input of 0 results in $\{0_1, 0_3, 0_4\}$. The same is true of inputs applied to $\{1_2\}$. These then are all the states that are needed. $\{0_1, 0_3, 0_4\}$ is the only state which has an output of 1. The completed state graph is therefore as in Fig. 5. Thus 6, and not 33, states result. The reduced state graph of this graph (see Moore[1]), by which the states $\{1_1\}$, $\{1_2\}$ and $\{0_2\}$ all collapse into a single state, has only 4 states!

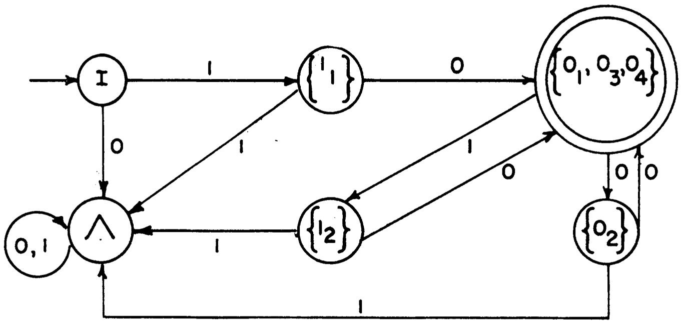

Fig. 5.

Assume that ϕ and Λ do not occur in the regular expression, since they can be eliminated by the following rules:

$$\phi = \sim((0\cup 1)(0\cup 1)^*),$$

$$\Lambda = \sim((0\cup 1)^*).$$

Actually, simplicity will often result by using the equations given at the end of section I wherever possible in eliminating ϕ and Λ; for example,

$$\alpha\Lambda = \Lambda.$$

Note first that if an expression has the tilde or cap in it, it is not possible to extend the method of section III in the obvious fashion. If a regular expression contains only cups, stars, and dots, at each digit of a sequence of 0's and 1's we can think of ourselves as being in any one of a set of positions of the regular expression. This seems to be no longer possible if the regular expression contains intersection or complementation.

The method for regular expressions containing a tilde or cap is roughly as follows. Suppose first that only a single tilde and no cap occurs in ρ, and that $\sim\alpha$ is a well formed part of ρ. A state graph for α is made into a state graph for $\sim\alpha$; then these states are used as positions along with the positions of the digits outside of $\sim\alpha$ in ρ in obtaining the state graph for ρ. If ρ contains a single cap and no tilde and $\alpha\cap\beta$ is a well formed part of ρ, then a state graph is obtained for $\alpha\cap\beta$ by a modification of the method of section III; using these states as positions as well as the digits of ρ outside $\alpha\cap\beta$, a state graph is obtained for ρ. If ρ contains several tildes or several caps or both tildes and caps, then the above steps must be repeated in the appropriate order. For example, if ρ is $\alpha\cdot((\sim\beta\cdot\sim\gamma)^*\cap\delta)$ where α, β, γ, and δ contain no caps or tildes, then in order to obtain a state graph for ρ, state graphs G_1 and G_2 are first obtained for $\sim\beta$ and $\sim\gamma$ respectively. Using states of G_1 and G_2 as positions, a state graph G_3 for $(\sim\beta\cdot\sim\gamma)^*$ is obtained. From G_3 and a state graph for δ, a state graph G_4 for $(\sim\beta\cdot\sim\gamma)^*\cap\delta$ is obtained. Then using the states of G_4 and the digits of α as positions, a state graph G is obtained for ρ.

IV. Complementation and Intersection in Regular Expressions

Because it is convenient to allow complementation and intersection in regular expressions that specify the behavior of automata, we extend the method of the previous section to the general class of regular expressions.

Theorem 4.1. If G is a state graph for α and G' results from G by reversing the output of every state of G from 0 to 1 or from 1 to 0, then G' is a state graph for $\sim\alpha$.

Proof: A sequence S is denoted by $\sim\alpha$ if and only if S is not denoted by α, which is so if and only if S applied to G results in output 0, which is so if and only if S applied to G' results in output 1. Q.E.D.

Theorem 4.2. Let G_1 and G_2 be state graphs for α and β, respectively. Let G be the state graph whose states are $\langle g_1 g_2 \rangle$ for all g_1 in G_1 and g_2 in G_2, such that there is a transition in G from $\langle g_1 g_2 \rangle$ to $\langle g_1' g_2' \rangle$ under input i if and only if there is a transition in G_1 from g_1 to g_1' under input i and a transition in G_2 from g_2 to g_2' under input i, such that the initial state of G is the state $\langle I_1 I_2 \rangle$ where I_1 and I_2 are the initial states of G_1 and G_2 respectively, and such that $\langle g_1 g_2 \rangle$ has output 1 in G if and only if g_1 has output 1 in G_1 and g_2 has output 1 in G_2. Then G is a state graph for $\alpha \cap \beta$.

Proof: A sequence S is denoted by $\alpha \cap \beta$ if and only if it is both denoted by α and denoted by β, which is so if and only if when S is applied to G_1 it yields an output 1 and when S is applied to G_2 it yields an output 1, which is so if and only if when S is applied to G there is an output 1 (which can be seen from the construction explicit in the statement of the Theorem). Q.E.D.

A state graph is *input-pure* if no two of its transitions leading into the same state have different inputs. For state graphs that are not input-pure, there is a simple method of constructing an equivalent input-pure state graph. For example, the state graph of Fig. 6, which is not input-pure can, by dividing S_3 into S_3' and S_3'', be converted into the equivalent state graph of Fig. 7, which is input-pure.

Let $\alpha_1, \cdots, \alpha_r$ be well-formed parts of ρ such that no α_i is a part of any other α_j, and such that no tilde or cap appears in ρ outside of the α's. For each i let G_i be an input-pure state graph for α_i. Then let the positions of ρ (with respect to $\alpha_1, \cdots, \alpha_r, G_1, \cdots, G_r$) be the digits of ρ outside of the α's and the states of $G_1, \cdots, G_r$. The digit of the latter type of position is the input on all the transitions leading to the state in the state graph. We must redefine some of the concepts introduced in section III.

A state g_i is an *initial* position of α_i if and only if there is a transition from the initial state of G_i to g_i in G_i. (Note that the initial state of G_i is not necessarily an initial position of α_i.) A state g_i of G_i is a *terminal* position of α_i if and only if g_i is a state of output 1 of G_i. From here on the definition of "initial" and "terminal" are the same as in section III. Well formed proper parts of the α's do not have initial and terminal positions, under the new definition, but all other well formed parts of ρ do.

$\langle P_1 P_2 \rangle$ is a transition of ρ (with respect to $\alpha_1, \cdots, \alpha_r, G_1, \cdots, G_r$) if and only if either 1) P_1 and P_2 are states of the same G_i and there is a transition in G_i from P_1 to P_2, or 2) P_1 and P_2 are not states of the same G_i and $\langle P_1 P_2 \rangle$ is a transition according to the definition of section III. An *allowable sequence of positions* of ρ (with respect to $\alpha_1, \cdots, \alpha_r, G_1, \cdots, G_r$) is a finite sequence of positions (with respect to $\alpha_1, \cdots, \alpha_r, G_1, \cdots, G_r$), $P_1, \cdots, P_n$, such that P_1 is an initial position of ρ, P_n is a terminal position of ρ and, for each $i<n$, $\langle P_i P_{i+1} \rangle$ is a transition of ρ. (This definition is the same, word for word, as that of section III, and the following theorem is almost the same as theorem 3.1.)

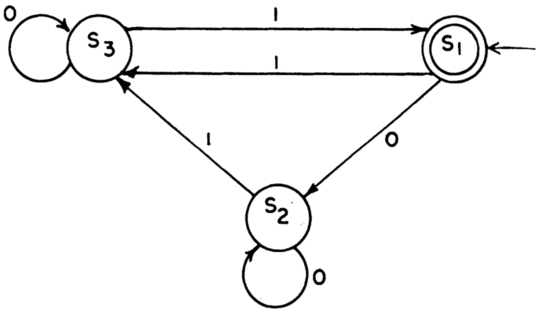

Fig. 6.

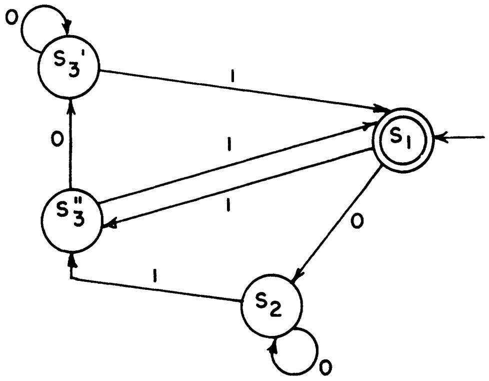

Fig. 7.

Theorem 4.3. If no tildes, caps, phis, or lambdas occur outside $\alpha_1, \cdots, \alpha_r$ in ρ, and if $G_1, \cdots, G_r$ are input-pure state graphs for $\alpha_1, \cdots, \alpha_r$, respectively, then a finite sequence S of 0's and 1's is denoted by a regular expression ρ if and only if there exists an allowable sequence R of positions for ρ (with respect to $\alpha_1, \cdots, \alpha_r, G_1, \cdots, G_r$), such that S results from R by replacing each position by the digit of that position.

The proof follows from the definitions. (As in the case of theorem 3.1 we allow the interested reader to construct the rigorous proof for himself.)

We close with an example. Let ρ be $0\sim[1(00\cup 01)^*0]11$. Let β be $[1(00\cup 01)^*0]$, and let α be $\sim\beta$. Then β is the ρ of the example at the end of section III. A state graph for β (obtained from the one of section III by reduction) is shown in Fig. 8. By converting single circles into double circles and vice versa, we obtain a state graph for α, which is then converted into the input-pure state graph G of Fig. 9. The positions of ρ with respect to α and G are 0_1, A, D_0, D_1, B_1, C, B_0, 1_1 and 1_2. (0_1, 1_1 and 1_2, of course, refer to positions of ρ outside α.) The initial positions of α are D_0 and B_1; the terminal posi-

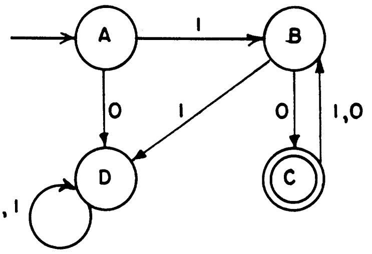

Fig. 8.

tions of α are all the states of G except C. The transitions of ρ are then

$\langle 0_1B_1\rangle$, $\langle 0_1D_0\rangle$

$\langle A1_1\rangle$

$\langle B_11_1\rangle$

$\langle B_01_1\rangle$

$\langle D_01_1\rangle$

$\langle D_11_1\rangle$

$\langle 1_11_2\rangle$,

and those shown on the state graph. Using the procedure of section III, we obtain the state graph of Fig. 10 for ρ. Note that the position A does not appear. In general the initial state s of G will not be used in obtaining the state graph for ρ if there is no transition into s in G, unless s is also an initial position in ρ. By the process of reduction, $\{B_0\}$ and $\{B_1\}$ collapse into a single state.

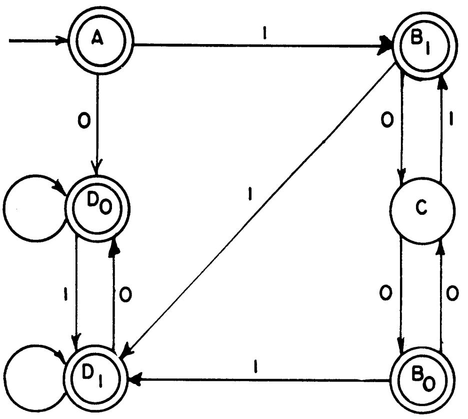

Fig. 9.

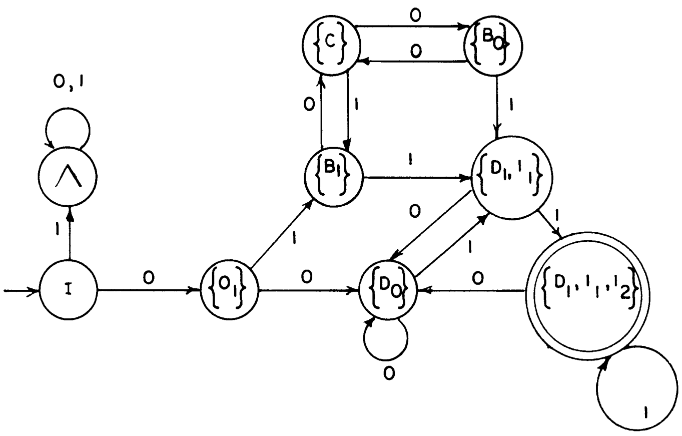

Fig. 10.