Link from markdown to pdf with rendered mathematical equations

Derivatives of Regular Expressions - Janusz A. Brzozowski (Modified)

Abstract

Kleene's regular expressions, were defined using three operators (union, concatenation and iterate) on languages. Word descriptions of problems can be more easily put in the regular expression language if the language is enriched by the inclusion of other logical operations. However, in the problem of converting the regular expression description to a state diagram, the existing methods either cannot handle expressions with additional operators, or are made quite complicated by the presence of such operators. In this paper the notion of a derivative of a regular expression is introduced and the properties of derivatives are discussed. This leads, in a very natural way, to the construction of a state diagram from a regular expression containing any number of logical operators.

1. Introduction

In this paper we use regular expressions, which can have any number of logical connectives, and describe methods for obtaining state diagrams from such regular expressions. We are concerned with: - the usual model of a finite automaton M (Moore model $\{11\}$). - the finite alphabet of inputs Σ, of size k, written as Σ_k. Hopefully this can be ignored: Each character can also be represented as binary 0 and 1. The n binary values $x_1, x_2, \ldots, x_n$ of a sequence are represented by a single 2^n valued input x, taking the values from $\Sigma_k = 0, 1, ..., k-1$, where $k = 2^n$. In other words, Σ_k represents all sequences up to length $log_2(k)$. - the internal states of M, $q_1, q_2, \ldots, q_m$ and one of these, q_lambda, is the starting state of M and finally - the transitions between states are specified by a flow table or by a state diagram.

2. Regular Expressions

A regular expression defines a language. We define the following operations on languages: If P and Q are two languages of symbols from our alphabet, Σ_k, we have:

Product or Concatenation. $(P.Q) = \{s | s = p.q; p \in P, q \in Q\}$. (Sometimes the dot is omitted for convenience. Also, since the operation is associative we omit parentheses.)

Iterate or Star Operation. $P^* = \cup_0^\infty P^n$, where $P^2 = P.P$, etc. and $P^0 = \lambda$, the set consisting of the string of zero length, which has the property $\lambda.R = R.\lambda = R$.

Boolean function. We shall denote any Boolean function of P and Q by $f(P, Q)$. The empty set is denoted by $\emptyset$ and the universal set by I. Thus we have the complement P' (with respect to I) of P, the *intersection* $P\&Q$, the sum or union $P + Q$, the modulo-two sum (exclusive OR) $P \oplus Q$, etc. Of course, all the laws of Boolean algebra apply. The operators: product, star and f are called regular operators. For economy of notation, we use the same symbol for a string s as for the language consisting of only that string s.

Definition 2.1. A regular expression is defined recursively as follows: 1. The symbols of the alphabet Σ_k, λ and $\emptyset$ are regular expressions. 2. If P and Q are regular expressions, then so are $P.Q$, P^*, and $f(P, Q)$, where f is any Boolean function of P and Q. 3. Nothing else is a regular expression unless its being so follows from a finite number of applications of Rules 1 and 2.

The definition is a modification of the definition given by McNaughton and Yamada {3}, but is more suitable for our purposes. The introduction of arbitrary Boolean functions enriches the language of regular expressions. For example, suppose we desire to represent the language of strings having three consecutive 1's but not those ending in 01 or consisting of 1's only. The desired expression is easily seen to be:

$$R = (I.1.1.1.I)\&(I.0.1 + 1.1^*)'.$$

TODO

- ☐ Define Inductive type for regular expressions as described
- ☐ Prove $\lambda.R = R.\lambda = R$.
- ☐ Write tests for $(I.1.1.1.I)\&(I.0.1 + 1.1^*)'$.

3. Derivatives of Regular Expressions

We define another operation on a languages R, yielding a new language called a derivative of R.

Definition 3.1. Given a language R and a finite string s, the derivative of R with respect to s is denoted by D_sR and is $D_sR = \{t | s.t \in R\}$.

The notion of derivative of a set (under different names) was introduced previously {10, 14, 15}, but was not applied to regular expressions. We now present an algorithm for finding derivatives of regular expressions. We shall need to know when a regular expression contains λ. For this purpose we make the following definition.

Definition 3.2. Given any language R we define $\delta(R)$ to be

$$\begin{aligned}\delta(R) &= \lambda \text{ if } \lambda \in R \\ &= \emptyset \text{ if } \lambda \notin R\end{aligned}$$

It is clear that:

$$\begin{aligned}&\delta(a) = \emptyset \text{ for any } a \in \Sigma_k, \\ &\delta(\lambda) = \lambda, \text{and} \\ &\delta(\emptyset) = \emptyset.\end{aligned}$$

Furthermore

$$\begin{aligned}\delta(P*) &= \lambda \text{ (by definition of } *), \text{ and} \\ \delta(P.Q) &= \delta(P) \;\&\; \delta(Q).\end{aligned}$$

If $R = f(P, Q)$ it is also easy to determine $\delta(R)$. For example,

$$\begin{aligned}&(3.1)\ \delta(P + Q) && = \delta(P) + \delta(Q). \\ &(3.2)\ \delta(P \;\&\; Q) && = \delta(P) \;\&\; \delta(Q). \\ &(3.3)\ \delta(P') && = \lambda \text{ if } \delta(P) = \emptyset \\ & && = \emptyset \text{ if } \delta(P) = \lambda\end{aligned}$$

where & and + is defined for δ similar to λ being True and $\emptyset$ being False in a boolean equation:

$$\begin{aligned}A \;\&\; B = \lambda &\text{ if and only if } A = \lambda \text{ and } B = \lambda \\ A + B = \emptyset &\text{ if and only if } A = \emptyset \text{ and } B = \emptyset\end{aligned}$$

The δ function of any other Boolean expression can be obtained using rules (3.1)-(3.3), since the connectives + and $'$ form a complete set of connectives.

THEOREM 3.1. If R is a regular expression, the derivative of R with respect to a character $a \in \Sigma_k$ is found recursively as follows:

$$\begin{aligned}&(3.4)\ D_a a && = \lambda, \\ &(3.5)\ D_a b && = \emptyset, \text{ for } b = \lambda \text{ or } b = \emptyset \text{ or } b \in A_k \text{ and } b \neq a, \\ &(3.6)\ D_a(P^*) && = (D_a P)P^*, \\ &(3.7)\ D_a(PQ) && = (D_a P)Q + \delta(P)(D_a Q). \\ &(3.8)\ D_a(f(P, Q)) && = f(D_a P, D_a Q).\end{aligned}$$

PROOF.

The proof follows for relations (3.4)-(3.8), for finding the derivative of R with respect to a character $a \in \Sigma_k$.

By definition 3.1, $D_aR = \{t|a.t \in R\}$. Then relations (3.4) and (3.5) are obvious. Thus the theorem holds for regular expressions involving no regular operators.

Let us consider now (3.8). It is sufficient to prove this relation for $f(P,Q) = P + Q$ and for $f(P,Q) = P'$, for this is a complete set of Boolean connectives. Now

$$\begin{aligned} D_a(P+Q) &= \{t|a.t \in (P+Q)\} \\ &= \{u|a.u \in P\} + \{v|a.v \in P\} \\ &= D_aP + D_aQ. \end{aligned}$$

It is clear that this rule can be extended to any number of regular expressions, i.e. that $D_a(R_1 + R_2 + \ldots) = D_aR_1 + D_aR_2 + \ldots$ even when the number of R_j is countably infinite. Next, note that $a.D_aR + a.D_aR' = a.I$. Taking the derivative with respect to a of both sides, we have $D_aR + D_aR' = I$. Also $(D_aR)\&(D_aR') = \emptyset$, and we have $D_aR' = (D_aR)'$. Thus rule (3.8) holds for union and complementation, and consequently for any Boolean function.

Next consider $D_aP.Q$. Let $P = \delta(P) + P_0$, where $\delta(P_0) = \emptyset$. Then

$$\begin{aligned} D_aPQ &= \{s|as \in (\delta(P) + P_0)Q\} \\ &= \{u|au \in \delta(P)Q\} + \{v|av \in P_0Q\} \\ &= \delta(P)(D_aQ) + \{v_1v_2|av_1 \in P_0, v_2 \in Q\} \\ &= \delta(P)(D_aQ) + \{v_1|av_1 \in P_0\}Q \\ &= \delta(P)(D_aQ) + (D_aP_0)Q. \end{aligned}$$

But $D_aP = D_a(P_0 + \lambda) = D_aP_0$; hence $D_a(PQ) = \delta(P)D_aQ + (D_aP)Q$, which is rule (3.7).

Finally we have

$$\begin{aligned} D_aP^* &= D_a(\lambda + P + PP + PPP + \ldots) \\ &= D_a\lambda + D_aP + D_aP^2 + \ldots D_aP^n \ldots. \end{aligned}$$

But

$$\begin{aligned} \sum_{n=1}^{\infty} D_aP^n &= \sum_{n=1}^{\infty}((D_aP)P^{n-1} + \delta(P)(D_aP^{n-1})) \\ &= \sum_{n=1}^{\infty}(D_aP)P^{n-1}, \end{aligned}$$

since $\delta(P)(D_a^{n-1})$ is either $\emptyset$ or it is $D_a P^{n-1}$, which is already included. Thus we have

$D_a P* = \sum_{n=1}^{\infty}(D_a P)P^{n-1} = (D_a P)\sum_{n=1}^{\infty} P^{n-1} = (D_a P)P*$,

which is rule (3.6). This concludes the proof of Theorem 3.1.

THEOREM 3.2. The derivative of a regular expression R with respect to a finite string of input characters $s = a_1, a_2, ..., a_m$ is found recursively as follows:

$$D_{a_1 a_2} R = D_{a_2}(D_{a_1} R),$$
$$D_{a_1 a_2 a_3} R = D_{a_3}(D_{a_1 a_2} R),$$
$$D_s R = D_{a_1 a_2 ... a_m} R = D_{a_m}(D_{a_1 a_2 ... a_{m-1}} R),$$

For completeness, if $s = \lambda$, then $D_\lambda R = R$. The proof follows from Definition 3.1.

TODO

- ☐ Can we define Definition 3.1 ?
- ☐ Define δ using Defintion 3.2
- ☐ See the proof in Appendix I and whether we should Define Theorem 3.1
- ☐ Proof Theorem 3.2 using Definition 3.1

4. Properties of Derivatives

It will be shown that the use of derivatives of a regular expression R leads to the construction of a state diagram of a sequential circuit characterized by R, in a very natural way. First, however, let us investigate some of the properties of derivatives.

THEOREM 4.1. The derivative $D_s R$ of any regular expression R with respect to any string s is a regular expression.

PROOF. It is clear from Theorem 3.2 that $D_\lambda R$ is regular and that $D_s R$ is regular (for s of length greater than 1) if $D_a R$ is regular (where a is a string of length 1). The construction of Theorem 3.1 shows that $D_a R$ is regular, since only a finite number of regular operations is required to find the derivative.

THEOREM 4.2. A string s is contained in a regular expression R if and only if λ is contained in $D_s R$.

PROOF. If $\lambda \in D_s R$, then $s.\lambda = s \in R$ from Definition 3.1. Conversely, if $s \in R$, then $s.\lambda \in R$ and $\lambda \in D_s R$, again from Definition 3.1.

Theorem 4.2 reduces the problem of testing whether a string s is contained in a regular expression R to the problem of testing whether λ is contained in $D_s R$. The latter problem is solved through the use of $\delta(D_s R)$.

Two regular expressions which are equal (but not necessarily identical in form) will be said to be of the same type.

Theorem 4.3.

- (a) Every regular expression R has a finite number d_R types of derivatives.
- (b) At least one derivative of each type must be found among the derivatives with respect to strings of length not exceeding $d_R - 1$.

PROOF. Part (a) of Theorem 4.3 is proved by induction on a number N of regular operators. BASIS, $N = 0$. The theorem is certainly true when R is one of $\emptyset$, λ or $a \in \Sigma_k$, for we have

$D_s\emptyset = \emptyset$, for all $s \in I$,

$D_\lambda\lambda = \lambda$, and

$D_s\lambda = \emptyset$, for all $s \in I$, $s \neq \lambda$.

$D_\lambda a = a$.

$D_a a = \lambda$.

$D_s a = \emptyset$ for all $s \in I$, $s \neq \lambda$, $s \neq a$.

Thus we have $d_\emptyset = 1$, $d_\lambda = 2$ and $d_a = 3$.

INDUCTION STEP, $N > 0$. Assume that each expression X with N or fewer operators has a finite number d_x of derivatives. If R is an expression with $N+1$ operators, there are three cases.

Case 1. $R = f(P, Q)$. It is easily verified from the definitions that $D_sR = D_s(P+Q) = D_sP + D_sQ$. Thus $d_R \leq d_P d_Q$. If $R = P'$ then $D_sR = (D_sP)'$. In this case, $d_R = d_P$. Since any Boolean function can be expressed using a finite number of sums and complements, it follows that the number of derivatives of R (of the form $R = f(P, Q)$) is finite.

Case 2. $R = P.Q$. Let $s = a_1a_2...a_m$. Using the definitions of Section 3, we have $D_{a_1}R = (D_{a_1}P)Q + \delta(P)D_{a_1}Q$. Similarly, for a string of length 2, we have $D_{a_1a_2}R = (D_{a_1a_2}P)Q + \delta(D_{a_1}P)D_{a_2}Q + \delta(P)D_{a_1a_2}Q$. In general, the derivative with respect to a string of length m will have the form

$$\begin{aligned}(\text{II.1})\ D_{a_1\ldots a_m}R = {} & (D_{a_1\ldots a_m}P)Q \\ & + \delta(D_{a_1\ldots a_{m-1}}P)D_{a_m}Q\ldots \\ & + \delta D_{a_1}P(\delta(D_{a_2\ldots a_m}Q) \\ & + \delta(P)(D_{a_1\ldots a_m}Q).\end{aligned}$$

Thus D_sR is the sum of $(D_sP)Q$ and of at most m derivatives of Q. If there are d_P and d_Q types of derivatives of P and Q respectively, there can be at most $d_R \leq d_P 2^{d_Q}$ types of derivatives of R. (Note that we are finding upper

bounds to show the finiteness of d_R, but such bounds do not necessarily have to be achieved.) Hence the inductive step holds for this case also.

Case 3. $R = P^*$. Again let us consider the formation of the derivative of P^*. We have

$$\begin{aligned} D_{a_1}(P^*) &= (D_{a_1}P)P^*, \\ D_{a_1a_2}(P^*) &= (D_{a_1a_2}P)P^* + \delta(D_{a_1}P)(D_{a_2}P^*) \\ &= (D_{a_1a_2}P)P^* + \delta(D_{a_1}P)(D_{a_2}P)P^*, \textit{etc.} \end{aligned}$$

It can be seen that, in general, D_sR will be the sum of terms of the form $D_t(P)P*$. If P has d_P types of derivatives, then R has at most $d_R \leq 2^{d_P} - 1$ types of derivatives. This concludes the inductive step.

PROOF. Part (b) of Theorem 4.3 indicates a method of finding all the d_R different types of derivatives, which will be called the characteristic derivatives of R. The strings from Σ_k can be arranged in order of increasing length; for example, $\Sigma_2 = \{0, 1\}$, so we have the following strings arranged in increasing length $\lambda, 0, 1, 00, 01, 10, 11, 000, \ldots$

The derivatives are now found in the above order, i.e. $D_\lambda R$, D_0R, D_1R, $D_{00}R$, $D_{01}R$, $D_{10}R$, $D_{11}R$, $D_{000}R$, ... If for strings s of length $L(s) = m$ no new types of derivatives are found, the process terminates. For, if no new derivatives are found for $L(s) = m$, then every D_sR is equal to (of the same type as) another derivative D_tR, where $L(t) < m$. Consider $D_{sa}R = D_a(D_sR) = D_a(D_tR) = D_{ta}R$, where $a \in \Sigma_k$ and $L(ta) < (m + 1)$. Thus every derivative with respect to a string sa of length $m + 1$ will be equal to some derivative with respect to a string ta of length $L(ta) < (m + 1)$. Hence, if no new types of derivatives are found for $L(s) = m$, then no new types will be found for $L(s) = m + l$, etc. Therefore at least one new type of derivative must be found for each $L(s)$ or, otherwise, the process terminates.

In the above discussion it is assumed that it is possible to decide when two derivatives are of the same type. This is not always an easy problem but it can be resolved, as is shown in Section 5.

THEOREM 4.4. Every regular expression R can be written in the form

$$(4.1)\quad R = \delta(R) + \sum_{a \in A_k} aD_aR,$$

where the terms in the sum are disjoint.

PROOF. First, R may or may not contain λ; this is taken care of by $\delta(R)$. If R contains a string s, then that, string must begin with a letter $a \in \Sigma_k$ of the input alphabet. In view of the definition of derivative, the set aD_aR is exactly language of R of strings beginning with a. The terms of the sum are obviously

disjoint, for the string in one term begin with a letter of Σ_k different from those in another term.

It follows from Theorem 4.4 that every regular expression can be represented by an infinite sum $R = \sum_{s \in I} sD_sR$. The number of types of derivatives is of course finite and so the series is redundant (e.g. the language of strings beginning with ab is contained in the language of strings beginning with a). The expansion (4.1) is much more useful, as will be shown below.

THEOREM 4.5. The relationship between the d_R characteristic derivatives of R can be represented by a unique set of d_R equations of the form

$$D_sR = \delta(D_sR) + \sum_{a \in A_k} aD_{u_a}R,$$

where D_sR is a characteristic derivative and $D_{u_a}R$ is a characteristic derivative equal to $D_{sa}R$. Such equations will be called the characteristic equations of R.

The theorem follows directly from Theorems 4.3 and 4.4.

THEOREM4.6. An equation of the form $X = A.X + B$, where $\lambda(A) = \emptyset$, has the solution $X = A^*B$, which is unique (up to equality of regular expressions).

The theorem is a modification of a theorem of Arden {8}, who has shown that $X = X.A + B$, $\delta(A) = \emptyset$, has the solution $X = BA^*$. The proof of Theorem 4.6 parallels that of Arden's theorem and will not be given here.

THEOREM 4.7. The set of characteristic equations of R can be solved for R uniquely: (up to equality).

PROOF. The proof follows from Theorem 4.6. The last characteristic derivative can be found in terms of the previous derivatives. Note that the coefficient A in the equation for any derivative is either $\emptyset$ or it is one or more symbols of the input alphabet. Thus $\delta(A) = \emptyset$ in all cases and Theorem 4.6 applies. Next, the solution for the last derivative can be substituted in the first $(d_R - 1)$ equations, reducing the number of equations by 1. The process is repeated until the set of equations is solved for $D_\lambda R = R$.

Thus the regular expression can be always reconstructed from the characteristic equations (although it may be in a different form, depending on the order of elimination of derivatives from the equation).

TODO

- ☐ Does our declaration of D's type assume too much by making THEOREM 4.1 too trivial?
- ☐ Prove THEOREM 4.2
- ☐ Prove Theorem 4.3 (a)
- ☐ Prove Theorem 4.3 (b)

☐ Prove Theorem 4.4
☐ Prove Theorem 4.5
☐ Prove Theorem 4.6
☐ Prove Theorem 4.7

5. State Diagram Construction

Definition 5.1. A string s is accepted by an automaton M (Moore Model $\{11\}$)with starting state q_λ iff when s is applied to M in q_λ the output is 1 at the end of s. Otherwise, s is rejected by M. A string s is accepted by a state q_j of M iff when M is started in q_j the output is 1 at the end of s.

Two states q_j and q_k of M are indistinguishable $\{11\}$ iff every string s applied to M started in q_j produces the same output string as that produced by applying s to M started in q_k.

THEOREM 5.1 $\{7\}$. Two states q_j and q_k of M are indistinguishable iff R_j and R_k denoting the languages of q_j and q_k are equal.

PROOF. (The theorem has been proved by Lee $\{7\}$ in a more general form.) If q_i and q_k are indistinguishable then M started in q_j and M started in q_k produce identical output string for any input string. In particular, the languages of strings ending in $Z = 1$ must be identical, i.e. $R_j = R_k$. Now suppose that $R_j = R_k$, but q_j and q_k are distinguishable. Then there must exist an s producing different output strings. Consider the first position in which the output strings differ; clearly, the initial portion of s up to and including that position is accepted in one case and rejected in the other, contradicting $R_j = R_k$.

It is clear that, if a string s takes M (which accepts R) from q_λ to q_j, then the regular expression R_j is simply the derivative of R with respect to s. If we take the first s that takes M from q_λ to q_j., then with each state we can associate a unique derivative with respect to that first s. Theorem 5.1 can now be restated as follows.

THEOREM 5.1 (a). Two states q_j and q_k of an automaton M characterized by the regular expression R are indistinguishable iff their derivatives are equal, i.e. $D_{s_j}R = D_{s_k}R$, where s_j and s_k are any two strings taking M from state q_λ to q_j and q_k respectively.

Thus by the arguments presented in this and the preceding sections we have established a very close relationship between derivatives of a regular expression and the states of the corresponding finite automaton. These results are summarized as follows:

1. To obtain a regular expression from a state diagram or a flow table, write the set of characteristic equations and solve for R. There is one equation for each state. If the transition under input a takes the graph from q_j to

q_k then the equation for R_j contains the term $a.R_k$; and λ is a term of R_j. if and only if the output for q_j is $Z = 1$.

There are other methods {3, 5, 8} of obtaining the regular expressions, but this otto is most closely related to the derivative approach.

2. To obtain the minimal state diagram from a regular expression, find the characteristic derivatives and associate one internal state with each characteristic derivative. The output associated with a state is $Z = 1$ iff the corresponding characteristic derivative contains λ.

This procedure is illustrated in the following example.

Let $R = (0 + 1)^*.1$

Then

$D_\lambda = R$; introduce q_λ with $Z = 0$, for $\delta(R) = \emptyset$,

$D_0 = R$; return q_λ under input 0,

$D_1 = R + \lambda$; introduce q_1, with output $Z = 1$ (for $\delta(D_1) = \lambda$), and a transition from q_λ to q_1 under input $x = 1$.

D_{00}, D_{01} need not be considered, for D_0 does not correspond to a new characteristic derivative, i.e. 0 returns the state graph to q_λ. Continuing, we find

$D_{10} = R$; go from q_1 to q_λ under $x = 0$,

$D_{11} = R + \lambda$; return from q_1 to q_1 under $x = 1$.

This completes the process. The resulting state diagram is shown in Figure 1(a).

$$q_{\lambda/0} \xrightarrow{0} q_{\lambda/0}$$
$$q_{\lambda/0} \xrightarrow{1} q_{1/1}$$
$$q_{1/1} \xrightarrow{0} q_{\lambda/0}$$
$$q_{1/1} \xrightarrow{1} q_{1/1}$$

Figure 1. Moore model state graph for $R = (0 + 1)^*.1$

In the above discussion we have assumed that, it is always possible to recognize the equality of two regular expressions. If this is the case, then the state diagram constructed by associating one internal state with each type of derivative is always minimal. However, it is often quite difficult to determine whether two regular expressions are equal. We now show that this difficulty can be overcome, and a state graph can always be constructed, but not necessarily with the minimum number of states. It should be pointed out that the other existing methods {3, 6} have the same difficulties and, moreover, are limited to regular expressions with (+), (.) and (*) only.

Definition 5.2. Two regular expressions are similar if one can be transformed to the other by using only the identities:

$$R + R = R,$$
$$P + Q = Q + P,$$
$$(P + Q) + R = P + (Q + R).$$

Two regular expressions are dissimilar iff they are not similar. It is clear that similarity implies equality, but equality in general does not imply similarity. Similarity can be easily recognized and is much weaker than equality.

THEOREM 5.2. Every regular expression has only a finite number of dissimilar derivatives.

PROOF.

To prove Theorem 5.2 we must demonstrate that the process of constructing the derivatives will terminate after a finite number of steps, even if only similarity of regular expression is recognized. This result is actually implicit in the proof of Theorem 4.3, but we shall explain it now in more detail.

The result is obvious for the basis step. Now suppose the given R is of the form $R = f(P, Q)$. Then from Case 1 above we have $D_sR = f(D_sP, D_sQ)$, for this result holds for $+$ and $'$. Now, as s takes on all possible values, compare D_sR with all the previously found derivatives. Since D_sP and D_sQ will appear in $f(D_sP, D_sQ)$ only a finite number of times and the structure of f is fixed, and furthermore, P and Q have a finite number of derivatives, the process will clearly terminate.

Proceeding with Case 2 above, if $R = P.Q$ we can write D_sR in the form of (II.1). Here, however we cannot use the argument of Case 1 since the number of terms in (II.1) increases with the length of s. The associative law for sum has been used in (II.1) to remove parentheses. The commutative law for sum can be used to recognize two sums in which the terms appear in different orders. But it is the identity $R+R = R$ which allows us to terminate the process. Note that the inequality $d_R \leq d_P 2^{d_Q}$ resulting from (II.1) does not depend on the length of s. Thus each new derivative must be simplified using $R + R = R$ and then compared with the previous derivatives.

Finally, the same argument is applicable to $R = P^*$, and hence the theorem holds.

As a consequence of this result, *a state diagram can be constructed even if only similarity among the derivatives is recognized*. However, such a method has a serious disadvantage since, in general, the diagram so constructed will be far from minimal. This arises because of the frequent appearance of λ and $\emptyset$ in the derivatives. For example, consider $R = (0 + 1)^*(01)$ and the derivative D_1R.

$$\begin{aligned} D_1R &= (D_1(0+1)^*).0.1 + (D_1 0.1) \\ &= ((D_1(0+1)).(0+1)^*).0.1 + (D_1 0).1 \\ &= ((D_1 0 + D_1 1)(0+1)^*)(0.1) + (D_1 0).1 \\ &= ((\emptyset + \lambda)(0+1)^*).(0.1) + \emptyset.1 \end{aligned}$$

In this case, using only similarity, we are forced to conclude that R and D_1R are dissimilar. However, the expression for D_1R can be easily simplified by the identities

$$\begin{aligned} &(5.1)\ R + \emptyset &&= \emptyset + R = R \\ &(5.2)\ R.\emptyset &&= \emptyset.R = \emptyset \\ &(5.3)\ R.\lambda &&= \lambda.R = R \end{aligned}$$

The expression for D_1R then becomes

$$\begin{aligned} D_1R &= ((\emptyset + \lambda)(0+1)*).(0.1) + \emptyset.1 \\ &= (0+1)^*.(0.1) + \emptyset \\ &= (0+1)^*.(0.1) \\ &= R \end{aligned}$$

The identities (5.1)-(5.3) are thus very useful and will be incorporated in the method.

We conclude this section with a more complicated example. Let it be desired to have an output if the input string contains two consecutive 0's but does not end it, 01. The required regular expression is $R = (I.0.0.I)\&(I.0.1)' = P\&Q'$, where $I = (0+1)^*$, $P = I.0.0.I$, $Q = I.0.1$. The construction of derivatives proceeds as follows:

$$\begin{aligned} D_\lambda &= R = P\&Q' && ;\text{introduce } q_\lambda, \\ D_0 &= (P + 0.I)\&(Q+1)' && ;\text{introduce } q_0, \\ D_1 &= P\&Q' && ;\text{introduce } q_1, \\ D_{00} &= (P + 0.I + I)\&(Q+1)' && ;\text{introduce } q_{00}. \end{aligned}$$

Here we note that $I + X = I$ and $I\&X = X$. Hence we may write D_{00} in a simpler form:

$$
\begin{aligned}
D_{00} &= (Q+1)', \\
D_{01} &= P\&(Q+\lambda)' && ;\text{introduce } q_{01}, \\
D_{000} &= (Q+1)' && ;\text{return to } q_{00}, \\
D_{001} &= (Q+\lambda)' && ;\text{introduce } q_{001}, \\
D_{010} &= (P+0.I)\&(Q+1)' && ;\text{return to } q_{0}, \\
D_{011} &= P\&Q' && ;\text{return to } q_{\lambda}, \\
D_{0010} &= (Q+1)' && ;\text{return to } q_{00}, \\
D_{0011} &= Q' && ;\text{introduce } q_{0011}, \\
D_{00110} &= (Q+1)' && ;\text{return to } q_{00}, \\
D_{00111} &= Q' && ;\text{return to } q_{0011}.
\end{aligned}
$$

This concludes the construction. The characteristic derivatives are D_λ, D_0, D_{00}, D_{01}, D_{001} and D_{0011}. Therefore the state diagram has 6 states. One has to determine $\delta(D_s)$ to find the output associated with q_s. In this case only D_{00} and D_{0011} contain λ; hence the output is $Z = 1$ only for q_{00} and q_{0011}.

The original paper has a figure 3, showing these 6 states, with their transitions, which was left out here, but I think it is easy to imagine or draw oneself.

Upon examining the state diagram it is seen that states q_λ q_{01} are indistinguishable and that the reduced state diagram contains only 5 states. We have failed to discover this because we have failed to recognize the equivalence

$D_{01} = P\&(Q+\lambda)' = P\&Q'\&\lambda' = P\&Q' = D_\lambda$

TODO

- ☐ Prove that $P\&Q'\&\lambda' = P\&Q'$
- ☐ Read section again and find more todos

7. Conclusion

Regular expressions can be obtained more easily from word description of problems if one is allowed to use any logical connective in the formation of the expression. We have introduced here the notion of a derivative of a regular expression as a powerful aid in analyzing the properties of regular expressions with arbitrary logical connectives.

Acknowledgment. The author wishes to thank Professor E. J. McCluskey, J. F. Poage, E. B. Eichelberger and S. O. Chagnon of Princeton University for their comments and suggestions.