6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

DFA of Non Unique Decodable Code

János Falucskai

Department of Mathematics and Informatics, College of Nyíregyháza
e-mail: falu@zeus.nyf.hu

Abstract

Sets of codewords can be represented by finite automata (FAs) and every FA can be represented by connection matrices or regular expressions. Our goal is to find similar systems and to solve one of the systems's problems in another system. Having a set of codewords our problem is to decide whether there is two or more sequences of codewords wich form the same chain of characters of codewords. (1 and 0 in binary case).

We show an algorithm (program) that solves this problem by using finite automata and their deterministic finite automata (DFAs).

1. Introduction

For example let the set of codes be $K = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, and $\alpha_1 = 01$, $\alpha_2 = 00$, $\alpha_3 = 011$, $\alpha_4 = 100$. If we have string $\beta = 01100$, we can see there are two different decompositions, namely $\beta = \alpha_3 \cdot \alpha_2 = \alpha_1 \cdot \alpha_4$. We show an algorithm solving this problem using finite automata and their deterministic finite automata (DFAs). We will get that non unique decodable strings are represented by DFA. There is another representation, in terms of regular expressions, which is often useful.

2. Analogies

In this section we list the analog systems using in area of examination, and show an example for all. The examples are for code $K = \{01, 00, 011, 100\}$. We have to consider that a codeword may be followed any codeword and sometime the chain has to end.

Left regular language:

$S \to B0|C0|C1|D1|\varepsilon$
$A \to S1$
$B \to A0$

$C \to S0$
$D \to C1$

Right regular language:

$S \to 1A|0C|\varepsilon$
$A \to 0B$
$B \to 0S$
$C \to 0S|1S|1D$
$D \to 1S$

Equation system of left regular expression:

$S = B0 + C0 + C1 + D1 + \varepsilon$
$A = S1$
$B = A0$
$C = S0$
$D = C1$

Equation system of right regular expression:

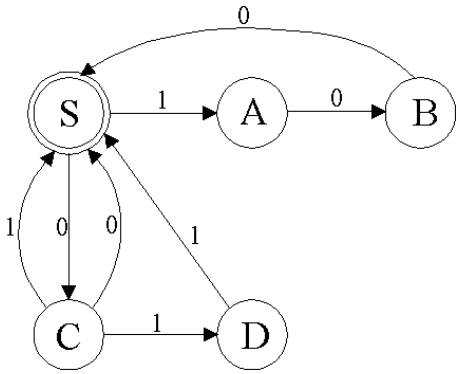
$S = 1A + 0C + \varepsilon$
$A = 0B$
$B = 0S$
$C = 0S + 1S + 1D$
$D = 1S$

Connection matrix:

	S	A	B	C	D
S	–	**1**	–	**0**	–
A	–	–	**0**	–	–
B	**0**	–	–	–	–
C	**0,1**	–	–	–	**1**
D	**1**	–	–	–	–

(In this case we have to give the startstate (S) and the set of finalstates $\{S\}$.)

Finite automaton:



Having a set of codewords our problem is to decide whether there is two or more sequences of codewords wich form the same chain of characters of codewords. We can see the following analogous systems: left and right regular languages, left and right regular expressions, finite automaton, connection matrices.

We can solve our problem in two ways: by using regular expressions or utilizing (deterministic) finite automaton.

3. Codes and Regular Expressions

In equations, "products" represent concatenation and are usually written without any explicit operator sign, and "sums" represent alternative choices and may be read as "or", or "union". The symbol ε represents an empty string.

A system of equations is "solved" in much the same way that a system of algebraic equations would be solved, any variable may substituted by an equivalent expression. The process of elimination would eventually leave a single variable defined in terms of constants, following which the whole chain of substitutions could be unravelled to obtain the values of all the variables.

There is only one technical point, which concerns the procedure to be followed when the same variable occurs on both sides of an equation, as in

$$X = Xa + b,$$

where X is unknown, a and b are known regular expressions.

3.1. The Arden's Lemma

In the next part we give a summary of solution of equation called Arden's lemma. First we have to define the "star" operator. Let A be a finite set, and A^* is the next expression:

$$\sum_{i=0}^{\infty} = A^*, \text{ where } A^i = A^* A^{i-1}$$

Since $A^i = A^* A^{i-1}$ is recursive formula, we have to give its expansion:
$A^0 = \varepsilon$
$A^1 = A$
* (products) represent concatenation.

We have four rules of inference:
- two expressions equal to a third are equal to each other
- an expression may be substituted for an equal expression
- the solution to $X = Xa + b$ is $X = ba^*$
- the solution to $X = aX + b$ is $X = a^*b$

These last two rules are the Arden's lemma. The notation a^* stands for the continued alternative $(\varepsilon + a + aa + aaa + \ldots)$ and is read "a star". Any symbolic expression constructed by the aid of concatenation, alternative selection, and the star operation, is called a *regular expression*. Regular expressions are ideally suited to describe paths through a finite automaton, and conversely, any regular expression has a diagrammatic representation.

If Σ is a finite set, ε denotes null string and $\emptyset$ is the empty set, we have the definition of a regular expression:
- ε is a regular expression
- $\emptyset$ is a regular expression
- $a \in \Sigma$ is a regular expression
- if x and y are regular expressions, so is xy
- if x and y are regular expressions, so is $x + y$
- if x is a regular expression, so is x^*

3.2. Systems of Equations

The regular expression's equation $X = \alpha X + \beta$ that can be solved by $X = \alpha^* \beta$ is just as applicable to systems of equations if they are written in matrix notation. Thus if we define the following vectors and matrices

$$X = \begin{pmatrix} A \\ B \end{pmatrix}, \ \alpha = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \beta = \begin{pmatrix} u \\ v \end{pmatrix}$$

we can write one single matrix equation:

$$X = \alpha X + \beta$$

whose solution would be

$$X = \alpha^* \beta$$

Of course, such a solution is not of much value unless there is an effective way to calculate the "star" of a matrix. Fortunately the matrix elements of the star are simple regular expressions of the elements of the matrix being starred, expressing in a concise form the results of the chain of substitutions that would otherwise be carried out explicitly each time a system of symbolic equations was to be solved.

Indeed, the easiest way to establish the point is to go ahead and solve the equations: Given

$$A = aA + bB + u$$

$$B = cA + dB + v$$

Disentangling u and v from the final equations for A and B yields a formula for α^*

$$\alpha^* = \begin{pmatrix} (a^*bd^*c)^*a^* & (a^*bd^*c)a^*bd^* \\ (d^*ca^*b)^*d^*ca^* & (d^*ca^*b)^*d^* \end{pmatrix},$$

Although these results have been stated for a 2×2 matrix, the fact that the matrix elements can be matrices themselves means that the results are valid in terms of submatrices. By repeated partitioning, in principle a system of any size can be reached. In practice, each partition multiplies the complexity of the symbolic expressions involved, quickly reaching unmanageable proportions. Alternative forms for these matrix elements can be derived using some of the identities satisfied by regular expressions, but the difficulty is a fundamental one, the formulas required are intrinsically complicated.

3.3. The Calculus of Regular Expressions

If we are genuinely interested in infinite solutions, we can entertain alternatives to Arden's lemma, but we should beware that the result may depend upon the node which we have chosen for the final equation; it will only express a solution involving that node. It often happens that a diagram contains a loop which makes a transition to a second loop, but that there is no return path to the first loop. If the equations are solved for a node in the first loop, the existence of the second loop will not be evident.

The algebra of regular expressions is not as direct as one would like, and often questions involving regular expressions are most easily resolved by constructing an equivalent diagram, transforming the diagram, and converting the result back into a regular expression. For example, regular expressions form a Boolean algebra, but the definition of a regular expression only involves unions - the regular expression sum. Intersections, which represent words conforming to several regular expression descriptions, and complements, which are words which in no way conform to the description, are readily defined in terms of diagrams, but not by operations directly on the regular expression itself. For example, excluded words are obtained through the subset construction, even though they are then readily described by regular expressions.

A related problem is the one of determining the equality of two regular expressions, and is the reason that there is no canonical form for regular expressions. Equality can be determined by constructing equivalent diagrams, reducing them to their simplest form and comparing the results; however there is no unique regular expression corresponding to any given diagram.

Although each automata can be described by a regular expression, the tendency is for the expression to become more and more complicated. Generally speaking,

the limit of a sequence of regular expressions need not be a regular expression. One of the charming aspects of this theory is its formal resemblace to the differential calculus of analysis. The derivative of Ω with respect to a is the set of tails of the words in Ω which begin with the letter a. Sets are the derivatives of sets; since regular expressions define sets it remains only. We use these methods to find not unique decodable strings of our codes by FAs, DFAs and regular expressions.

3.4. How to Find the Not Unique Decodable Strings

First, we can define a finite automaton (FA) according to the code. Our next step is to make this automaton deterministic (DFA), and define the regular equation system of DFA. We get the not unique decodable string(s) by solving the equation system -using Arden's lemma if necessary- and omitting the whole codewords from the result. For example:

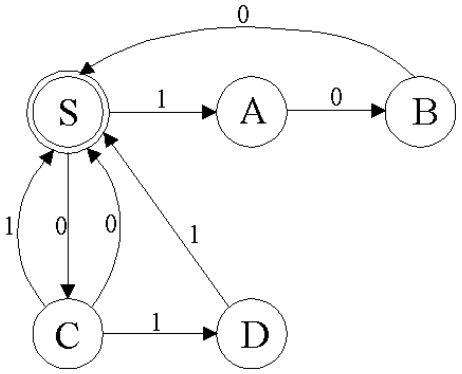


Figure 1: The FA of code $K = \{01, 00, 011, 100\}$

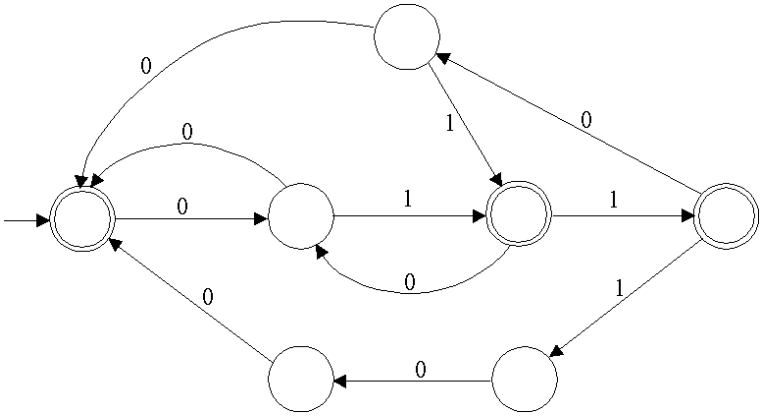


Figure 2: The DFA of code $K = \{01, 00, 011, 100\}$

Regular expression of DFA is the following: $0(00)^*1(01)^*1\{(011)^*00 + 100\} = 0\alpha_2^*1\ \alpha_1^*1\{\alpha_3^*00 + 4\}$, if we omit those code words wich are assigned by *, we get the string: 01100.

4. Code and DFA, about the Program

We can get answer for existing of the not unique decodable string during construction of DFA. If we have two or more final state in one new state involving by the same symbol, then at this new state two or more codewords will end. Since our automaton is deterministic, we found a chain which is closed by different codewords, so this chain is a not unique decodable string.

Our application use this method to find the answer. The program has developed in Visual Basic 6.0. It is able to determine FA and DFA for a code, and can describe the automata in graphical form or as a special text file: first row of the text file contains the final states and the remaining rows contain the rules of automaton.

References

[1] Janusz A. Brzozowski: Derivatives of regular expressions, Journal of the Association for Computing Machinery (1964) 481-494.

[2] Conway, J.H, Regular Algebra and Finite Machines, Chapman, Ltd., London, 1971 (ISBN 412-10620-5).

[3] Mc Intosh, H: Wolfram's class IV automaton and a good life. Physica D 45 (1990), 45-105.

Postal address

János Falucskai
Department of Mathematics and Informatics
College of Nyíregyháza
4400 Nyíregyháza, Sóstói út 31/B
Hungary