Brzozowski Algebraic Method

Michael Levet

April 18, 2015

1 Introduction

Recall that a language is regular if and only if it is accepted by some finite state automaton. In other words, each regular expression has a corresponding finite state automaton. An important problem in Automata Theory deals with converting between regular expressions and finite state automata. The Brzozowski Algebraic Method is an intuitive algorithm which takes a finite state automaton and returns a regular expression. This algorithm relies on notions from graph theory and linear algebra; particularly, graph walks and the substitution method for solving systems of equations.

2 Notions From Graph Theory

In this section, the notions of a graph walk and the adjacency matrix will be introduced, along with some relevant results. I start with the definition of a walk.

Walk: Let G be a graph, and let $v = (v_1, v_2, ..., v_n)$ be a sequence of vertices (not necessarily distinct). Then v is a walk if v_i, v_{i+1} are adjacent for all $i \in \{1, ..., n-1\}$.

Intuitively, we start at some vertex v_1. Then we visit v_2, which is a neighbor of v_1. Then v_3 is a neighbor of v_2. Consider the cycle graph on four vertices, C_4, which is shown below. Some example walks include (v_a, v_b, v_d) and $(v_a, v_b, v_a, v_c, v_a)$. However, (v_a, v_d, v_a, v_b) is not a walk, as v_a and v_d are not adjacent.

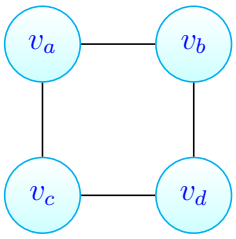
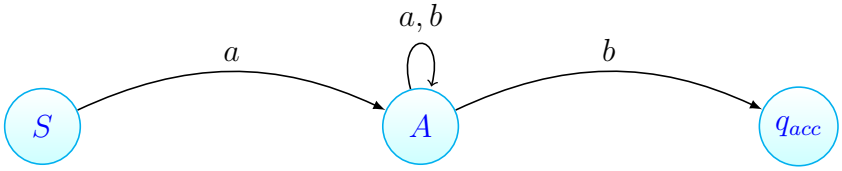


Figure 1: The graph C_4.

Consider an example of a walk on a directed graph. Observe that (S, A, A, q_{acc}) is a walk, but (S, A, S) is not a walk as there is no directed edge (A, S) in the graph.



The adjacency matrix will now be defined, and we will explore how it relates to the notion of a walk.

Adjacency Matrix: Let G be a graph. The adjacency matrix $A \in \{0,1\}^{V \times V}$, with $A_{ij} = 1$ if vertex i is adjacent to vertex j, and $A_{ij} = 0$ otherwise.

Note: If G is undirected, then $A_{ij} = A_{ji}$. Finite state automata diagrams are directed graphs, though, so A_{ij} may be different than A_{ij}.

Consider the simple C_4 graph above. The adjacency matrix for this graph is:

$$A(C_4) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Similarly, the adjacency matrix of the above directed graph is:

$$A(G) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

The adjacency matrix is connected to the notion of a walk by the fact that $(A^n)_{ij}$ counts the number of walks of length n starting at vertex i and ending at vertex j. The proof of this theorem provides the setup for the Brzozowski Algebraic Method. For this reason, I will provide a formal proof of this theorem.

Theorem: Let G be a graph and let A be its adjacency matrix. Let $n \in \mathbb{N}$. Then each cell of A^n, denoted $(A^n)_{ij}$, counts the number of walks of length n starting at vertex i and ending at vertex j.

Proof: This theorem will be proven by induction on $n \in \mathbb{N}$. Consider the base case of $n = 1$. By definition of the adjacency matrix, if vertices i and j are adjacent, then $A_{ij} = 1$. Otherwise, $A_{ij} = 0$. Thus, A counts the number of walks of length 1 in the graph. Thus, the theorem holds at $n = 1$.

Suppose the theorem holds for an arbitrary integer $k > 1$. The theorem will be proven true for the $k + 1$ case. As matrix multiplication is associative, $A^{k+1} = A^k \cdot A$. By the inductive hypothesis, A^k counts the number of walks of length k in the graph, and A counts the number walks of length 1 in the graph. Consider:

$$(A^{k+1})_{ij} = \sum_{x=1}^{n} ((A^k)_{ix} \cdot A_{xj})$$

And so for each $x \in \{1, ..., n\}$, $(A^k)_{ix} \cdot A_{xj} \neq 0$ if and only if there exists at least one walk of length k from vertex i to vertex x, and an edge from vertex k to vertex j. Thus, the theorem holds by the principle of mathematical induction.

Example: Consider again the graph C_4, and $(A(C_4))^2$, given below. This counts the number of walks of length 2 on G. Observe that $((A(C_4))^2)_{14}$ states that there are two walks of length 2 from vertex v_a to vertex v_d. These walks are (v_a, v_b, v_d) and (v_a, v_c, v_d).

$$(A(C_4))^2 = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 2 & 0 & 0 & 2 \end{bmatrix}$$

3 Brzozowski Algebraic Method

The Brzozowski Algebraic Method takes a finite state automata diagram (the directed graph) and constructs a system of linear equations to solve. Solving a subset of these equations will yield the regular expression for the finite state automata. I begin by defining some notation. Let E_i denote the regular expression which takes the finite state automata from state q_0 to state q_i.

The system of equations consists of recursive definitions for each E_i, where the recursive definition consists of sums of $E_j R_{ji}$ products, where R_{ji} is a regular expression consisting of the union of single characters. That is, R_{ji} represents the selection of single transitiosn from state j to state i, or single edges (j, i) in the graph. So if $\delta(q_j, a) = \delta(q_j, b) = q_i$, then $R_{ji} = (a + b)$. In other words, E_j takes the finite state automata from state q_0 to q_j. Then R_{ji} is a regular expression describing strings that will take the finite state automata from state j to state i in exactly one step. That is:

$$E_i = \sum_{j \in Q, \text{there exists a walk from state j to state i}} E_j R_{ji}$$

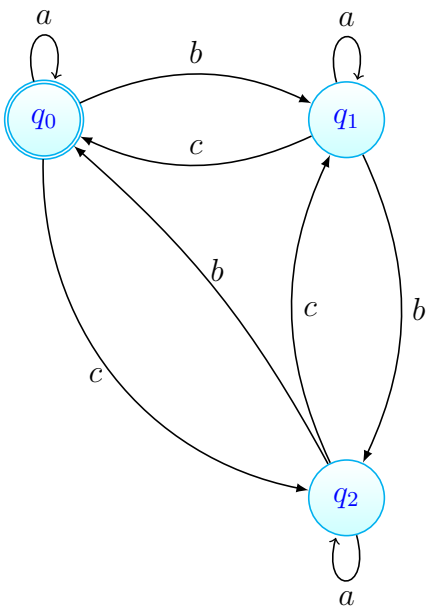
Note: Recall that addition when dealing with regular expressions is the set union operation.

Once we have the system of equations, then we solve them by backwards substitution just as in linear algebra and high school algebra.

The explanation of this algorithm is dense, though. Let's work through an example to better understand it. We seek a regular expression over the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ describing those integers whose value is 0 modulo 3.

In order to construct the finite state automata for this language, we take advantage of the fact that a number $n \equiv 0 \pmod 3$ if and only if the sum of n's digits are also divisible by 3. For example, we know $3|123$ because $1 + 2 + 3 = 6$, a multiple of 3. However, 125 is not divisible by 3 because $1 + 2 + 5 = 8$ is not a multiple of 3.

Now for simplicity, let's partition Σ into its equivalence classes $a = \{0, 3, 6, 9\}$ (values congruent to 0 mod 3), $b = \{1, 4, 7\}$ (values equivalent to 1 mod 3), and $c = \{2, 5, 8\}$ (values equivalent to 2 mod 3). Similarly, we let state q_0 represent a, state q_1 represent b, and state q_2 represent c. Thus, the finite state automata diagram is given below, with q_0 as the accepting halt state:



We consider the system of equations given by E_i, taking the FSM from state q_0 to q_i:

- $E_0 = \lambda + E_0a + E_1c + E_2b$

  If at q_0, transition to q_0 if we read in the empty string, or if we go from $q_0 \to q_0$ and read in a character in a; or if we go from $q_0 \to q_2$ and read in a character in c; or if we go from $q_0 \to q_2$ and read in a character from b.

- $E_1 = E_0b + E_1a + E_2c$

  To transition from $q_0 \to q_1$, we can go from $q_0 \to q_0$ and read in a character from b; go from $q_0 \to q_1$ and read in a character from a; or go from $q_0 \to q_2$ and read in a character from c.

- $E_2 = E_0c + E_1b + E_2a$

  To transition from $q_0 \to q_2$, we can go from $q_0 \to q_0$ and read a character from c; go from $q_0 \to q_0$ and read in a character from b; or go from $q_0 \to q_2$ and read in a character from a.

Since q_0 is the accepting halt state, only a closed form expression of E_0 is needed.

There are two steps which are employed. The first is to simplify a single equation, then to backwards substitute into a different equation. We repeat this process until we have the desired closed-form solution for the relevant E_i (in this case, just E_0). In order to simplify a variable, we apply Arden's Lemma, which states that $E = \alpha + E\beta = \alpha(\beta)^*$, where α, β are regular expressions.

We start by simplifying E_2 using Arden's Lemma: $E_2 = (E_0c + E_1b)a^*$.

We then substitute E_2 into E_1, giving us $E_1 = E_0b+E_1a+(E_0c+E_1b)(a)^*c = E_0(b+ca^*c)+E_1(c+ba^*c)$. By Arden's Lemma, we get $E_1 = E_0(b + ca^*c)(a + ba^*c)^*$

Substituting again, $E_0 = \lambda + E_0a + E_0(b + ca^*c)(a + ba^*c)^*c + (E_0c + E_1b)a^*b$.

Expanding out, we get $E_0 = \lambda + E_0a + E_0(b + ca^*c)(a + ba^*c)^*c + E_0ca^*b + E_0(b + ca^*c)(a + ba^*c)^*a^*b$.

Then factoring out: $E_0 = \lambda + E_0(a + ca^*b + (b + ca^*c)(a + ba^*c)^*(c + ba^*b))$.

By Arden's Lemma, we have: $E_0 = (a + ca^*b + (b + ca^*c)(a + ba^*c)^*(c + ba^*b))^*$, a closed form regular expression for the integers mod 0 over Σ.

Note: The hard part of this algorithm is the careful bookkeeping. In a substitution step, the regular expression for an E_i may grow and require simplification. Be careful to keep track of how the regular expression is expanded on a substitution step, and how it is possible to factor out terms.