Converting Deterministic Finite Automata to Regular Expressions

Christoph Neumann

Mar 16, 2005

Abstract

This paper explores three techniques for converting deterministic finite automata (DFA) to regular expressions and compares the usefulness of each technique. The techniques examined are the transitive closure method, the state removal method, and the Brzozowski algebraic method.

1 Background

Kleene's seminal article defines regular expressions and their relationship to finite automata [7]. Kleene proves the equivalence of finite automata and regular expressions thereby providing us with the first technique, the transitive closure method, for converting DFAs to regular expressions. Later Brzozowski expanded on Kleene's method by introducing the notion of derivatives of regular expressions [3], but his paper passed into obscurity until G. Berry and R. Sethi brought Brzozowski's paper to the forefront in [2][1]. The state removal method appears in [4] but Linz presents a more straightforward method in [8].

2 Definitions

We will using the Moore model [9] for finite automata. Given an automaton M with an input alphabet $A_k = \{0, 1, \cdots, k-1\}$, M has m states $M_s = \{q_1, q_2, \cdots, q_m\}$ where q_λ is the starting state of M and $M_f = \{q_{f_1}, q_{f_2}, \cdots, q_{f_n}\}$ are the n final states of M where $n \leq m$. For clarity, we will use letters to represent each value in the alphabet A instead of using the numeric representation ($a = 0, b = 1, etc$), and for convenience we will assume q_1 is the starting state unless otherwise noted.

We will use Kleene's definition [7] of regular expressions. Regular expressions are defined recursively as:

1. The symbols $0, 1, \cdots, k-1, \lambda,$ and ϕ are regular expressions, where λ is the empty string and ϕ is the empty set.

[1]However, efficiently converting regular expressions to automata is the focus of the G. Berry and R. Sethi paper, not converting DFAs to regular expressions

2. *Set Union*: Given the regular expressions x and y, the union of x and y, expressed as $x + y$, is a regular expression.
3. *Concatenation*: Given the regular expressions x and y, the concatenation (or product) of x and y, expressed as xy is a regular expression.
4. *Iteration*: Given the regular expression x, the iteration (or star) of x, expressed as $x*$, is a regular expression.
5. Given a regular expression x, (x) is a regular expression.
6. All regular expressions can be constructed by a finite application of rules 1-5.

The regular expressions in rule 1 are terminals. The concatenation of terminals is a string. For a given regular expression x, there is a set X which contains all the strings represented by x. We will use x and X interchangeably. A single string is simply represented by the set containing that single string.

We will also refer to and use the following identities:

$$\begin{aligned} (ab)c = a(bc) &= abc \\ \lambda x = x\lambda &= x \\ \phi x = x\phi &= \phi \\ \phi + x &= x \\ \lambda + x* &= x* \\ (\lambda + x)* &= x* \end{aligned}$$

Since + is commutative, all the commutative versions apply.

3 Transitive Closure Method

Suppose the given DFA M is to be represented as a regular expression.

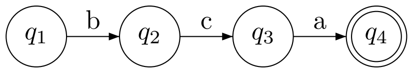

Figure 1: A very simple automaton.

Consider the automaton in figure 1. The input for edge in the automaton is a regular expression. Quite simply, the regular expression for the transition from q_1 to q_2 is b, the transition from q_2 to q_3 is c and so on. Furthermore, the regular expression representing the transition from q_1 to q_3 is the concatenation of the regular expressions thus forming bc. Thusly, we can find the regular expression for the automaton to be bca since that expression is the concatenation of all of the transitions from the starting state q_1 to the final state q_4.

More generally, for a path from q_λ to q_f, the concatenation of the regular expression for each transition in the path forms a regular expression that represents the same string as the path from q_λ to q_f in the automaton. Supposing there exists only one unique path in automaton M from

q_λ to q_f, there exists only one regular expression R such that R represents the same string as the DFA M. However, this is a trivial automaton, let us examine how to expand this to a more general case.

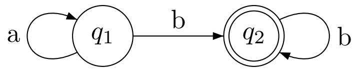

Figure 2: A more complicated automaton.

Now consider the DFA in figure 2. It is clear that multiple paths exist from q_1 to q_2. We cannot derive a simple regular expression to represent the DFA, however using the other operators (union and iteration) we can build on our previous approach to create a construction that works for all types of DFA. We will forgo the proof and leave it to Kleene [7]. The proof is also demonstrated by Hopcroft and Ullman [5] and explained quite clearly by Salomaa [10].

Suppose regular expression R_{ij} represents the set of all strings that transition the automaton M from q_i to q_j. Furthermore, suppose R_{ij}^k represents the set of all strings that transition the automaton M from q_i to q_j without passing through any state higher than q_k. We can construct R_{ij} by successively constructing $R_{ij}^1, R_{ij}^2, \cdots, R_{ij}^m$. R_{ij}^k is recursively defined as:

$$R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})*R_{kj}^{k-1} + R_{ij}^{k-1}$$

assuming we have initialized R_{ij}^0 to be:

$$R_{ij}^0 = \begin{cases} r & \text{if } i \neq j \text{ and } r \text{ transitions } M \text{ from } q_i \text{ to } q_j \\ r + \lambda & \text{if } i = j \text{ and } r \text{ transitions } M \text{ from } q_i \text{ to } q_j \\ \phi & \text{otherwise} \end{cases}$$

As we can see, this successive construction builds up regular expressions until we have R_{ij}. We can then construct a regular expression representing M as the union of all $R_{\lambda f}$ where q_λ is the starting state and $f \in M_f$ (the final states for M).

This technique is similar in nature to the all-pairs shortest path problem. The only difference being that we are taking the union and concatenation of regular expressions instead of summing up distances. This solution is of the same form as transitive closure and belongs to the constellation of problems associated with closed semirings.

The chief problem of the transitive closure approach is that it creates very large regular expressions. Examining the formula for an R_{ij}^k, it is clear the significant length is due to the repeated union of concatenated terms. Even by using the previous identities, we still have long expressions.

4 State Removal Method

The state removal approach identifies patterns within the graph and removes states, building up regular expressions along each transition. The advantage of this technique over the transitive closure method is that it is easier to visualize. This technique is described by Du and Ko [4], but we examine a much simpler approach is given by Linz [8].

First, any multi-edges are unified into a single edge that contains the union of inputs. Suppose from q_2 to q_5 there is an edge a and an edge b, those would be unified into one edge from q_2 to q_5 that has the value $a + b$.

Now, consider a subgraph of the automaton M which is of the form given in figure 3. State q may be removed and the automaton may be reduced to the form in figure 4. The pattern may still be applied if edges are missing. For an edge that is missing, leave out the corresponding edge in figure 4.

This process repeats until the automaton is of the form in figure 5. Then by direct calculation, the regular expression is:

$$r = r_1 * r_2 (r_4 + r_3 r_1 * r_2) * \quad .$$

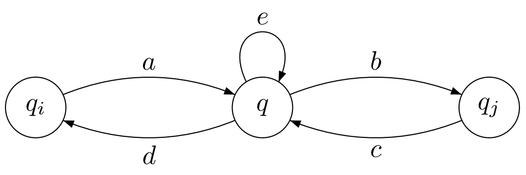

Figure 3: Desired pattern for state removal.

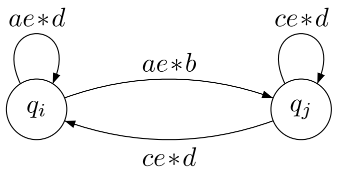

Figure 4: Results after state removal.

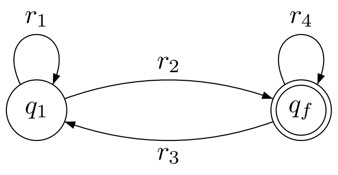

Figure 5: Final form.

5 Brzozowski Algebraic Method

Brzozowski method [3][2] takes a unique approach to generating regular expressions. We create a system of regular expressions with one regular expression unknown for each state in M, and then we solve the system for R_λ where R_λ is the regular expression associated with starting state q_λ. These equations are the characteristic equations of M.

Constructing the characteristic equations is straightforward. For each state q_i in M, the equation for R_i is a union of terms. Each term can be constructed like so: for a transition a from q_i to q_j, the term is aR_j. If R_i is a final state, λ is also one of the terms. This leads to a system of equations in the form:

$$\begin{aligned} R_1 &= a_1R_1 + a_2R_2 + \cdots \\ R_2 &= a_1R_1 + a_2R_2 + \cdots \\ R_3 &= a_1R_1 + a_2R_2 + \cdots + \lambda \\ \vdots &= \vdots \\ R_m &= a_1R_1 + a_2R_2 + \cdots + \lambda \end{aligned}$$

where $a_x = \phi$ if there is no transition from R_i to R_j.

The system can be solved via straightforward substitution, except when an unknown appears on both the right and left hand side of the equation. This situation occurs when there is a self loop for state q_i. Arden's theorem [1] is the key to solving these situations. The theorem is as follows:

> Given an equation of the form $X = AX + B$ where $\lambda \notin A$, the equation has the solution $X = A{*}B$.

We use this equation to isolate R_i on the left hand size and successively substitute R_i into the another equation. We repeat the process until we have found R_λ with no unknowns on the right hand side.

For example, consider again the automaton in figure 2. The characteristic equations are as follows (where $R_\lambda = R_1$):

$$\begin{aligned} R_1 &= aR_1 + bR_2 \\ R_2 &= bR_2 + \lambda \end{aligned}$$

We solve for R_2 using Arden's theorem and the previously mentioned identities:

$$\begin{aligned} R_2 &= bR_2 + \lambda \\ &= b{*}\lambda \\ &= b{*} \end{aligned}$$

We substitute into R_1 and solve:

$$\begin{aligned} R_1 &= aR_1 + b(b{*}) \\ &= aR_1 + bb{*} \\ &= a{*}(bb{*}) \\ &= a{*}bb{*} \end{aligned}$$

Thus, the regular expression for the automaton in figure 2 is $a{*}bb{*}$.

[2]Kain [6] explains this method in more detail and gives illustrative examples.

6 Conclusions

The state removal approach seems useful for determining regular expressions via manual inspection, but is not as straightforward to implement as the transitive closure approach and the algebraic approach. The transitive closure approach gives a clear and simple implementation, but tends to create very long regular expressions. The algebraic approach is elegant, leans toward a recursive approach, and generates reasonably compact regular expressions. Brzozowski's method is particularly suited for recursion oriented languages, such as functional languages, where the transitive closure approach would be cumbersome to implement.

References

[1] Dean N. Arden. Delayed-logic and finite-state machines. In *Theory of Computing Machine Design*, pages 1–35. U. of Michigan Press, Ann Arbor, MI, 1960.

[2] G. Berry and R. Sethi. From regular expressions to deterministic automata. *TCS: Theoretical Computer Science*, 48:117–126, 1987.

[3] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[4] Ding-Shu Du and Ker-I Ko. *Problem Solving in Automata, Languages, and Complexity*. John Wiley & Sons, New York, NY, 2001.

[5] John E. Hopcroft and Jeffery D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, Reading, MA, 1979.

[6] Richard Y. Kain. *Automata Theory: Machines and Languages*. Robert E. Krieger Publishing Company, Malabar, FL, 1981.

[7] S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata studies*, pages 3–40. Ann. of Math. Studies No. 34, Princeton University Press, Princeton, NJ, 1956.

[8] Peter Linz. *An introduction to Formal Languages and Automata*. Jones and Bartlett Publishers, Sudbury, MA, third edition, 2001.

[9] E. F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pages 129–153. Ann. of Math. Studies No. 34, Princeton University Press, Princeton, NJ, 1956.

[10] Arto Salomaa. *Jewels of Formal Language Theory*. Computer Science Press, Rockville, MD, 1984.