Converting DFA to Regular Expression Notes

Let's start with a DFA $M = (Q, \Sigma, \Delta, S, F)$

states: Q; alphabet: Σ; transition function: Δ; start state: S; Ending State: F

We can write an equivalent regular expression using the following function:

$$R(p, q, X)$$

which returns a regular expression for all the strings that get us from state p to state q going only through states in X where X is a potentially equal subset of Q. Note that p and q don't have to be in X.

First, let's think about the base case: $X = \emptyset$. Here, this function evaluates to:

If $p = q$: all the characters that transition us from p to p in one step, plus the empty string.

$$\begin{cases} a_1 + \dots + a_k + \varepsilon & \text{if } k \geq 1 \\ \varepsilon & \text{if } k = 0 \end{cases}$$

where $a_1 \dots a_k$ are all symbols such that $(p, a_i, q) \in \Delta$

If $p \neq q$: all the characters that transition us from p to q in one step

$$\begin{cases} a_1 + \dots + a_k & \text{if } k \geq 1 \\ \emptyset & \text{if } k = 0 \end{cases}$$

Okay, that's great. But what if $X \neq \emptyset$? Then, this function evaluates to:

$$R(p,q,X) = R(p,q,X \setminus \{v\}) + R(p,v,X \setminus \{v\})\left(R(v,v,X \setminus \{v\})\right)^* R(v,q,X \setminus \{v\})$$

where V is a state in Q.

But what does this mean?

$$\underbrace{R(p,q,X)}_{} = \underbrace{R(p,q,X\setminus\{v\})}_{} + \underbrace{R(p,v,X\setminus\{v\})}_{}\underbrace{(R(v,v,X\setminus\{v\})}_{})^* \underbrace{R(v,p,X\setminus\{v\})}_{}$$

the RegEx that gets us to q from p going through states in X.

The RegEx that gets us to q from p going though states in X that aren't v.

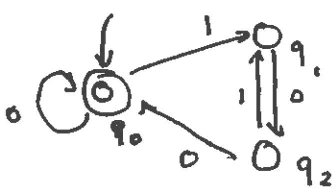
The RegEx that gets us from p to v through states in X that aren't v.

The RegEx that gets us from v to v using any state in X that isn't v.

The RegEx that gets us from v to p using any state in X that isn't v.

In other words, to get from p to q, we can either go from p to q without going through v, or we can go from p to v (without going through v first) loop back to v any number of times, and then go from v to p without looping through v again, and, we can write the RegEx that gets us from p to q as the regex that gets us from p to q without going through v, or the regex that goes from p to v, loops through v any number of times, and then the regex that gets us from v to q.

Okay, so that's a lot of words. Let's work through an example



okay, so to start we have

$$R(q_0, q_0, \{q_0, q_1, q_2\})$$

Now, let's break this down (I could choose any state as v, and I'll get different, but equivalent, regular expressions)

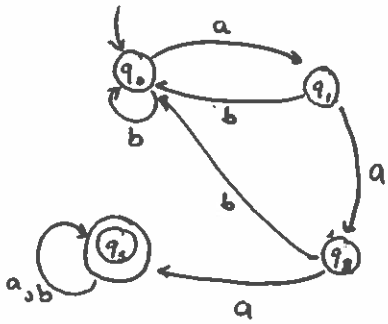
(here $V = q_1$)

$$= \underbrace{R(q_0, q_0, \{q_0, q_2\})}_{0^*} + \underbrace{R(q_0, q_1, \{q_0, q_2\})}_{0^*1}(R(q_1, q_1, \{q_0, q_2\}))^* R(q_1, q_0, \{q_0, q_2\})$$

$$0^* \quad + \quad 0^*1 \quad (00^*1)^* \quad 000^*$$

so this simplifies to $0^* + 0^*1(00^*1)^*000^*$

Example 2:



$$R(q_0, q_3, \{q_0, q_1, q_2, q_3\})$$

$$= R(q_0, q_3, \{q_0, q_2, q_3\}) + R(q_0, q_1, \{q_0, q_2, q_3\})(R(q_1, q_1, \{q_0, q_2, q_3\}))^* R(q_1, q_3, \{q_0, q_2, q_3\})$$

there's no way to get from $q_0 \to q_3$ without going through q_1, so $\emptyset$

$$= \underbrace{R(q_0, q_1, \{q_0, q_2, q_3\})}_{b^* a} \underbrace{(R(q_1, q_1, \{q_0, q_2, q_3\}))^*}_{(ab^*a)^*} \underbrace{R(q_1, q_3, \{q_0, q_2, q_3\})}_{aa(a+b)^*}$$

$$= b^* a (ab^* a)^* aa (a+b)^*$$

(Now, in some cases, you might have to expand this out further, but it gets really, really long very quickly.

Also, if we look at the DFA, we can see that this dfa accepts $(a+b)^* aaa (a+b)^*$, so the reg ex's we get are equivalent but not very human readable!