CSE 322 - Introduction to Formal Methods in Computer Science
Regular Operations on Languages

Dave Bacon
Department of Computer Science & Engineering, University of Washington

I. A QUESTION

Now that we have defined deterministic finite automata, we can begin to delve into the properties of these devilish little machines. Lets begin this delving by looking at a very simple question. Suppose that you have the following two DFAs:

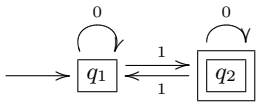

FIG. 1: Machine M_1

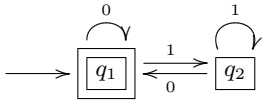

FIG. 2: Machine M_2

A little bit of thought will show that the language of machine M_1 is

$$A = L(M_1) = \{w | w \text{ has an odd number of 1s}\} \tag{1}$$

and the language of machine M_2 is

$$B = L(M_2) = \{w | w \text{ is the empty string } \varepsilon \text{ or } w \text{ ends in } 0\} \tag{2}$$

Since A and B are both generated by a finite automata, these are both regular languages. But now suppose that we want to produce a finite automata that accepts strings which begin with elements from A and end with those from B. Is it possible to construct a finite automata which does this? I'll let you ponder this question, as it isn't as easy as you might think at first sight. Indeed it will lead us down an interesting new path in the world of finite automata: nondeterministic finite automata. But more on that soon.

II. REGULAR OPERATIONS

Okay so it seems that the question we are asking is something like, given some languages, what new languages can we construct from these languages which are regular languages (can be generated by a DFA.) We will study a set of operations called *regular operations.* Why are they called that? You'll see in a second (patience young grasshopper.) Okay what follows are a list of definitions for some regular operations. These are not all regular operations, by the way, but an important class of such operations.

Let A and B be languages (sets of strings.) Then we define the *union* of A and B as the new language

$$A \cup B = \{w | w \in A \text{ or } w \in B\} \tag{3}$$

Thus the union of A and B is a new language which includes all of the strings which are in A and all of the strings which are in B.

Similarly we can define the *intersection* of A and B as the new langage

$$A \cap B = \{w | w \in A \text{ and } w \in B\} \tag{4}$$

Thus the intersection of A and B is a new language which includes all of the strings that are in both A and B.

We define the *concatenation* of A and B as the new language

$$A \circ B = \{wv | w \in A \text{ and } v \in B\} \tag{5}$$

In other words, the concatenation of the two languages A and B is made up of strings which come first from w and then from v. Note that $A \circ B$ is not equal to $B \circ A$ (the $\circ$ operator is not *commutative*.)

We define the *star* of a language A as the new language

$$A^* = \{x_1 x_2 \dots x_k | k \geq 0 \text{ and each } x_i \in A\} \tag{6}$$

Okay having defined these operations are languages, we can now ask an interesting question. Do these operations preserve the concept of a regular language? In other words if the languages used in these definitions can be generated by a DFA, can the the newly defined construction using the above operations also be generated by a DFA?

III. THE UNION OF REGULAR LANGUAGES IS A REGULAR LANGUAGE

So let's tackle the first of these questions. Let's ask whether the union of two regular languages is regular. We will show that indeed, the union of two regular languages is regular. How will we do this? We will do this by construction.

Let A and B be regular languages. Because they are regular we know that there is some DFA which recognizes these languages. Call these M_1 and M_2, such that $L(M_1) = A$ and $L(M_2) = B$. We will now show how to construct a new DFA M such that $L(M) = A \cup B$. This will prove that the union of two regular languages is a regular language.

So how do we construct M from M_1 and M_2? Think about this for a bit, and then read on.

We will construct M by *simulating* the action of M_1 and M_2, *at the same time*. Intuitively the picture is as follows. To simulate both M_1 and M_2 at the same time, we need to keep track of which of the states each of these DFA is in, and then update this state as each incoming character is read. At the end of the computation, we should then accept if either of the two machines is in the accept state. Okay, so this is the intuition, but the method we have just described isn't obviously a DFA. For example, we now seem to be keeping track of two states in our DFA. But if machine M_1 has states Q_1 and machine M_2 has states Q_2, then in order to keep track for the states of two machines, we can do this by keeping track of an ordered pair of elements (q_1, q_2), where $q_1 \in Q_1$ and $q_2 \in Q_2$. In other words, keeping track of two DFAs is equivalent to keeping track of a DFA whose state space is $Q_1 \times Q_2$.

This leads us to the formal construction of M:

Let M_1 recognize A where $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ and let M_2 recognize B where $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$. For simplicity, lets assume that $\Sigma_1 = \Sigma_2$. We will now show how to construct a DFA M which recognizes $A \cup B$. Then our new M has the following 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$:

1. $Q = Q_1 \times Q_2$. In other words $Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
2. $\Sigma = \Sigma_1 = \Sigma_2$. Here we have assumed that the alphabets of both machines are the same. If the machines have different alphabets, then we can still construct an M, but we will ignore this case for now.
3. The transition function, δ is constructed as follows. Recall that it will be a function of the form $\delta : Q \times \Sigma \to Q$, which for our definitions is $\delta : (Q_1 \times Q_2) \times \Sigma \to (Q_1 \times Q_2)$. Intuitively the transition function should update each part of $Q_1 \times Q_2$ separately. We can do this by setting

   $$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \tag{7}$$

   Thus the transition function gets as input the two states and the next letter in the string, and outputs the next state of each of the two machines.
4. The start state starts M in the initial states of the two machines: $q_0 = (q_1, q_2)$.
5. $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. This expression says that the accept states are made up of states where one of the two machines is in the

Now informally, we can see how this construction of M leads to a DFA which accepts a string when it is in A or B. A formal proof of this, however, would need to apply the definition of how a DFA computes to show that M acts as desired.

Note also that we can easily modify this proof to show that the intersection of two regular languages is regular. All we need to do is modify the accept states. Instead of what we have above, we would replace the accept states by $F = F_1 \otimes F_2$. Then the machine will accept only when both are accepting, which is the definition of the intersection.

IV. AN EXAMPLE

Let's see an example of the union construction for the automata given in Figures 1 and 2. Since both of these automata have only two states q_1 and q_2, we know that the machine which accepts the union of the two languages will be $Q = \{(q_1, q_1), (q_1, q_2), (q_2, q_1), (q_2, q_2)\}$. Given this insight we can then easily construct the state diagram by looking at the action M_1 and M_2 would take on taking as input different elements of the alphabet.

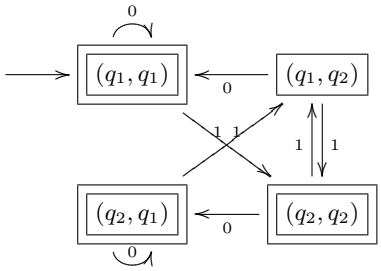

FIG. 3: Machine M