CSE 322 - Introduction to Formal Methods in Computer Science
Converting DFAs to Regular Expressions

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

Last time we saw how, given a regular expression, we could take this expression and construct an NFA which accepted the language of this regular expression. Thus if a language is expressible by a regular expression, the language is regular. Now we will show that if a language is regular, then the language is expressible by a regular expression. We will do this by showing that a DFA can be converted into a regular expression whose language is the language of the DFA. In order to achieve this result we will introduce yet another type of machine, a GNFA, where the G stands for *generalized*. You can bet this will be among the last times were modify the finite automata model, because how can you go beyond generalizing? (Colonels are below Generals, eh?)

I. GENERALIZED NONDETERMINISTIC FINITE AUTOMATA

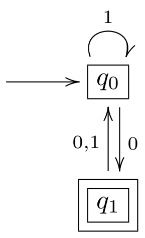
Generalized nondeterministic finite automata (GNFA) are nondeterministic finite automata in which instead of having transitions labeled by sets of elements of the alphabet, the transitions are labeled by regular expressions. A GNFA works by reading blocks of symbols from the input, not just one input at a time, according to the regular expression labeling a transition.

We will work with GNFAs which satisfy a list of requirements which will make our life easier. This list includes

1. The start state has no arrows coming into it and has an out arrow leading to every other state in the GNFA.
2. There is only one accept state. This accept state has no arrows coming out of it, but has an arrow coming into it from every other state in the GNFA.
3. For all other states besides the start state and accept state, there are transitions between all of these states as well as from each of these states to themselves.

To begin our discussion of how to convert a DFA into a regular expression, we first start out by converting a DFA into a GNFA. This can always be done by constructing a GNFA with two extra states, the new start and accept states of the GNFA and properly modifying the transitions in the GNFA. In particular we take the original DFA and add a start state and an accept state. From this new start state we add arrows to all other states. All of these new transitions are labeled by $\emptyset$ unless the transition is to the original start state of the DFA. For the transition to the original start state we label the transition by ε. Next we take all states in the original DFA and add arrows from these states to the new accept states. All of these transitions are labeled by $\emptyset$, except for transitions from the accept states of the original DFA, which are labeled by $\emptyset$. Finally for the connections among the states of the original DFA, we make two modifications. If there is not a transition between two states or from a state to itself, we add this transition and label it by $\emptyset$. If a transition between two states is labeled by more than one arrow, we label this state by the regular expression giving the union of these symbols.

Lets do this for a very simple example. Suppose our DFA is



Then the GNFA corresponding to this DFA is given by

$$\begin{tikzcd}
\boxed{q_s} \arrow[r, "\varepsilon"] \arrow[rr, bend left, "\emptyset"] \arrow[dr, "\emptyset"] & \boxed{q_0} \arrow[loop above, "1"] \arrow[r, "\emptyset"] \arrow[d, shift left, "0"] & \boxed{\boxed{q_a}} \\
& \boxed{q_1} \arrow[u, shift left, "0\cup 1"] \arrow[loop below, "\emptyset"] \arrow[ur, "\varepsilon"] &
\end{tikzcd}$$

.

So we have seen how we can convert a DFA into a GNFA which is two states bigger than the DFA. In order to show how we can obtain a regular expression from a DFA, we will now show how we can take a $k > 2$ state GNFA and convert it into a $k - 1$ state GNFA. If we do this repeatedly starting from the GNFA we constructed from our DFA, we will wind up with a GNFA which has 2 states. Such a GFNA will consist of a start state and an accept state with a transition labeled by a regular expression. This regular expression will be the regular expression corresponding to the DFA.

II. TURNING A k STATE GNFA INTO A $k - 1$ STATE GNFA

We will now (informally at first) discuss how to take a k state GNFA with $k > 2$ and convert it into a $k - 1$ state GNFA. We will do this by selecting one of the states in the GNFA which is not the start or accept state and removing it from the GNFA. When we remove this state will will have to repair the GNFA so that it still accepts the same language. If we do this repeatedly starting from a k state GNFA we will eventually end up with a 2 state GNFA.

How do we do this removing of a state from the GNFA? Well supose that we pick a state q_r which we are removing from the GNFA. What we will do is that for every incoming and outgoing transition for the GNFA (including ones which come in and return to the same state) we will apply a procedure which successfully repairs the removing of q_r. To illustrate this it is useful to examine the diagram of a pair of states, one with an incoming transition, and one with an outgoing transition,

$$\begin{tikzcd}
\boxed{q_i} \arrow[rr, "r_4"] \arrow[dr, "r_1"] & & \boxed{q_j} \\
& \boxed{q_r} \arrow[ur, "r_3"] \arrow[loop below, "r_2"] &
\end{tikzcd}$$

Here q_r is the state we are ripping out. q_i and q_j are here shown to be different, but they could be the same. In order to successfully repair the GNFA when we remove q_r we need to remove the arrows shown and replace them with one arrow from q_i to q_j which has a regular expression of

$$\begin{tikzcd}
\boxed{q_i} \arrow[r, "r_1 r_2^* r_3 \cup r_4"] & \boxed{q_j}
\end{tikzcd}$$

Those letters are small but if you squint you will see that they read $r_1 r_2^* r_3 \cup r_4$. Intuitively, what is going on is $r_1 r_2^* r_3$ is taking care of paths in the GNFA which go through q_r starting in q_i and ending in q_j. Since the state could have made this transition by itself, we also need to include the fact that it could go via the transition labeled by r_4. Thus we need to union $r_1 r_2^* r_3$ and r_4.

III. LET'S GET FORMAL

Okay, lets get formal on this problem. First of all we need to define GNFAs formally. A GNFA is similar to a NFA with the exception that the transition function is of a particular form and is now labeled by regular expression.

Let q_s and q_a be the start and accept states of the GNFA. Then the transition function will be a function from $(Q - \{q_a\}) \times (Q - \{q_s\})$ to the set of regular expressions over the alphabet we are using, Σ. Note that the transition function we are going to define this way is different from the way we have normally defined transition function (which took as input states and a symbol or set of symbols from the alphabet.) Here if we have $\delta(q_i, q_j) = r$, then we mean that the transition from q_i to q_j is labeled by the regular expression r. The domains we have given above guarantee that our GNFA is of the form we described above.

Okay, so here is the real formal definition. A GNFA is given by the 5-tuple $(Q, \Sigma, \delta, q_s, q_a)$ where

1. Q is the finite set of states which includes the states q_s and q_a.
2. Σ is our friend the alphabet.
3. The transition function is given by $\delta : \{Q - \{q_s\}\} \times \{Q - \{q_a\}\} \rightarrow R$ where R is the set of all regular expressions over the alphabet Σ.
4. The start state is q_s.
5. The accept state is q_a.

Given this definition, we can then write down the rules form when a GNFA accepts a string w in Σ^*. We say that a string $w \in \Sigma^*$ is accepted if $w = w_1 w_2 \dots w_k$ where each w_i is in Σ^* and a sequence of states $q_0, q_1, \dots, q_k$ exists such that

1. q_0 is the start state: $q_0 = q_s$.
2. q_k is the accept state: $q_k = q_a$.
3. For each i we have that the substring w_i is in the language of the regular expression r, $w_i \in L(r_i)$, and r_i is the regular expression of the transition, $\delta(q_{i-1}, q_i) = r_i$.

Now lets formally describe the process of removing states from the GNFA. Let $G = (Q, \Sigma, \delta, q_s, q_a)$ be a GNFA with $k > 2$ states. Define the procedure REDUCE(G) as follows. The procedure will take as input the GNFA G and produce as output a GNFA given by $G' = (Q', \Sigma, \delta', q_s, q_a)$. The new states Q' is produced by randomly selecting any state of Q which is not q_s and q_a and removing it from Q. Call this state q_r, then $Q' = Q - \{q_r\}$. To define the new transition function, we repair all of the arrows as described above. Formally: for any $q_i \in Q' - \{q_a\}$ and $q_j \in Q' - \{q_s\}$ define the new transition function by

$$\delta'(q_i, q_j) = \delta(q_i, q_r)\delta(q_r, q_r)^*\delta(q_r, q_j) \cup \delta(q_i, q_j)$$

. Using REDUCE(G) we can define another procedure CONVERT(G) which takes as input a $k \geq 2$ state GNFA and returns the regular expression whose language is the same as GNFA.

CONVERT(G) :

1. Let $k = |Q|$ for the GNFA $G = (Q, \Sigma, \delta, q_s, q_a)$.
2. If $k = 2$, then there is a single transition arrow between q_s and q_a. Output this regular expression. In other words output $r = \delta(q_s, q_a)$.
3. If $k > 2$ then apply REDUCE(G) producing a new GNFA G'. Return CONVERT(G').

Note that this is a recursive definition.

Now lets prove that the regular expression returned by CONVERT(G) is equivalent to G. We will do this by induction on the number of states in G.

Basis: If $k = 2$, then the GNFA is made up of only 2 states and one transition function from this state to the accept state. The regular expression label for this this transition describes all strings that allow G to get to the accept state. Hence this expression is equivalent to G.

Inductive step: To do this, we need to show that our procedure REDUCE(G) which produces G' and G accept the same languages. Suppose that G accepts the input w. Then in an accepting branch of the computation, G enters into the state $q_s, q_1, q_2, \dots, q_a$. If none of these states is the state we remove, then clearly G' will accept the string. This is because the newly constructed G' will only access transitions which have not been changed in our construction of G'. If, on the other hand q_r does appear, then the states, q_i and q_j, between which q_r appears in G' have a regular expression accepting all strings taking

q_i to q_j. Thus G' must accept w. Now we must also show that if G' accepts w then G accepts w in our construction of REDUCE(G). Each transition arrow in G' describes either a transition which was taken directly in G or via q_r in G. Thus G must also accept this string. Thus we have shown that REDUCE(G) and G accept the same language. The inductive hypothesis tells us that when the algorithm calls itself recursively on input G' the result is a regular expression equivalent to G' because G' has $k-1$ states. Hence, as we have just shown, the regular expression is equivalent to G.

IV. EXAMPLE

Lets work through an example. Indeed lets do it for the DFA we considered above. As we noted above the GNFA constructed directly from this DFA has four states:

$$\begin{tikzcd}
\boxed{q_s} \arrow[r, "\varepsilon"] \arrow[rr, bend left, "\emptyset"] \arrow[dr, "\emptyset"'] & \boxed{q_0} \arrow[r, "\emptyset"] \arrow[loop above, "1"] \arrow[d, shift left, "0"] & \boxed{\boxed{q_a}} \\
& \boxed{q_1} \arrow[u, shift left, "0\cup 1"] \arrow[ur, "\varepsilon"'] \arrow[loop below, "\emptyset"] &
\end{tikzcd}$$

Lets begin by removing q_0. This will create a regular expression transition from q_s to q_a given by $\varepsilon 1^*\emptyset \cup \emptyset$. Since concatenating with an empty set produces and empty set, this is equivalent to the $\emptyset$. Second, there will be a regular expression transition from q_1 to itself, the one created by traversing through q_0. This one is given by $(0 \cup 1)1^*0 \cup \emptyset$. We can reduce this to $\Sigma 1^*0$. There is a transition from the start state to q_1. This has the regular expression, $\varepsilon 1^*0 \cup \emptyset = 1^*0$. Finally, there is a transition from q_1 to the accept state q_a with regular expression $(0 \cup 1)1^*\emptyset \cup \varepsilon = \varepsilon$. The resulting new GNFA with 3 states is therefore

$$\begin{tikzcd}
\boxed{q_s} \arrow[rr, bend left, "\emptyset"] \arrow[dr, "1^*0"'] & & \boxed{\boxed{q_a}} \\
& \boxed{q_1} \arrow[ur, "\varepsilon"'] \arrow[loop below, "\Sigma 1^*0"] &
\end{tikzcd}$$

.

Next we apply the same procedure to the three state GNFA. This will yield only a transition between the start and accept states. The regular expression for this transition will be $1^*0(\Sigma 1^*0)^*\varepsilon \cup \emptyset$. This can be reduced to $1^*0(\Sigma 1^*0)^*$. The final GNFA is

$$\begin{tikzcd}
\boxed{q_s} \arrow[r, "1^*0(\Sigma 1^*0)^*"] & \boxed{\boxed{q_a}}
\end{tikzcd}$$

. Thus the regular expression for this DFA is $1^*0(\Sigma 1^*0)^*$. Cool.