Converting a DFA to a REG EXP

Exposition by William Gasarch

1 The Basic Algorithm

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. We can assume $Q = \{1, 2, \ldots, n\}$. We show how to construct a reg expression α that generates the same set the DFA M recognizes.

Let $R(i, j, k)$ be a regular expression for the set of strings x such that if you run M started at state i, only using states $\{1, \ldots, k\}$ (or a subset of them), you end up in state j.

We first show how to find $R(i, j, 0)$. Then, assuming one has $R(i, j, k-1)$ for ALL i, j, we derive $R(i, j, k)$ for ALL i, j.

$R(i, j, 0)$: Note that the only way to NOT use ANY states as intermediaries is to either transition directly from i to j. Hence the following seems reasonable:

$$R(i, j, 0) = \{\sigma \in \Sigma \mid \delta(i, \sigma) = j\}.$$

This IS correct if $i \neq j$. However, if $i = j$ then the empty string also takes you from state i to state i without using any intermediary states. So

$$R(i, i, 0) = \{e\} \cup \{\sigma \in \Sigma \mid \delta(i, \sigma) = j\}.$$

(NOTE: To understand this next equation you really need to be in class.)

$$R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1)R(k, k, k-1)^* R(k, j, k-1)$$

Hence, by induction on k, all of the $R(i, j, k)$ are regular expressions.

Assume that the start state is 1. The regular expression we seek is

$$\bigcup_{f \in F} R(1, f, n)$$

2 Two Tricks to Speed it Up

Given an n-state DFA it seems like one needs to computer $R(i, j, k)$ for all $1 \leq i, j \leq n$, $0 \leq k \leq n$, which is $n^2(n-1)$ different $R(i, j, k)$. But do we need all of them?

TRICK ONE: lets fine ot which ones we need. We need all $R(1, f, n)$ for $f \in F$ (assuming 1 is the start state). One can work backwards from there. For example, if we need $R(1, 9, 29)$ then we need

$R(1, 29, 28)$, $R(29, 29, 28)^*$ and $R(29, 9, 28)$.

We can keep working like this to get exactly which ones we need. THEN we just computer those. It may be that past some point computing which ones you need is itself difficult, so then you may want to just compute the rest yourself. So you may compute only some of the $R(-, -, 29)$, $R(-, -, 28)$, ..., $R(-, -, 10)$ but then do all of the $R(-, -, [0-9])$.

TRICK TWO: Let q be a state such that, for all $\sigma \in \Sigma$, $\delta(q, \sigma) = \sigma$. ($q$ may or may not be final or starting.) Then going from q to q you never need to use intermediary states. Hence

$$R(q, q, n) = R(q, q, 0).$$

You can skip all of the $R(q, q, i)$ calculations.

TRICK THREE: If p, q are states such that there is NO directed path from p to q, then $R(p, q, i) = \emptyset$ for all i.

COMBINE TRICK ONE WITH TRICKS TWO,THREE: When trying to determine which $R(i, j, k)$ you need (TRICK ONE), lets say you come across a case of either TRICK TWO or TRICK THREE. So say we fine we need $R(q, q, i)$ where q is a trap state. Then we DO NOT see what we need to computer this, we just KNOW its $R(q, q, 0)$. We may note that we need $R(q, q, 0)$ but we DO NOT need (for example) $R(q, q, i - 1)$. Or if we want that we need $R(p, q, i)$ and p and q are not connected, then we KNOW its $\emptyset$ and DO NOT bother with seeing what we need compute it.